Geometry and Reconstruction of Bipartite Quantum Correlations

by

Jan Bosma

To obtain the degree of Bachelor of Science in Applied Mathematics and Applied Physics at the Delft University of Technology

Student number:	4667093
Supervisors:	Dr. D. de Laat (EEMCS)
	Dr. S. Groeblacher (TNW)
Other committee members:	Prof. Dr. Ir. A.W. Heemink (EEMCS)
	Dr. D. Elkouss Coronas (TNW)

Delft, August 2020

ABSTRACT

The first part of this thesis provides a mathematical description for bipartite quantum correlations, aiming to analyze the geometry of several sets of correlations. We explain why quantum entanglement can be used to simulate shared randomness: $C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma)$ for a sufficiently large d. The known bound for this dimension d in the literature is $d \geq \dim(C_{loc}(\Gamma)) + 1$, but we improve this by showing that the inclusion is always true for $d \geq \dim(C_{loc}(\Gamma))$. For the proof of this bound, we show that the set $C_{\text{private}}(\Gamma)$ of correlations using private randomness is connected, which allows the use of an improved version of Carathéodory's Theorem. In the second part of this thesis, we define and analyze a see-saw method to determine the state and measurement operators that reconstruct both the correlation itself as its entanglement dimension, by solving consecutive semidefinite programs. One of the strengths of the algorithm is its generality: it applies to different dimensions, question sets, and answer sets. Some numerical experiments demonstrated that the method can indeed reconstruct quantum correlations, although some highly entangled correlations failed to be reconstructed due to the computational limitations. The numerical experiments motivated several new theorems, for example the fact that every correlation with $|A| = 1$ or $|B| = 1$ has entanglement dimension 1, which means that it can be written as a private randomness correlation. The proof of this result is based on the earlier described improvement for the dimension d.

CONTENTS

1

INTRODUCTION

Quantum mechanics. May I ask, how does it make you feel? Baffled? Impressed? Challenged? Fascinated? Personally, whenever I see it, a warm grin appears on my face. Quantum mechanics always teases me with its complexity, its hidden behaviour, and its perplexing properties. However, as confusing as quantum mechanics may be, it appears to be even more promising! Recent research shows that it has innumerable applications: from the *supremacy* of quantum computers [Aru+19], to a vision for quantum internet [WEH18]. Also, Bell's hypothesis [Bel64] regarding *entanglement* was recently successfully tested [Hen+15], suggesting quantum-nonlocality ("spooky action at a distance"); an instantaneous and invisible connection between separated particles. Curiously, this experiment was performed in Delft, only a few meters from my house!

In this thesis, the concept of a *bipartite correlation* is used to investigate how we can *reconstruct* a bipartite quantum correlation. In other words, we seek ways to efficiently make any correlation that you wish. Also, we are interested in determining the amount of entanglement necessary to construct a correlation. This is a nontrivial task, as this can not be directly measured in a lab. Nowadays, if you want to determine the entanglement dimension, you must know some a priori properties of the system (such as the density matrix), which are not always available. And even if they are, you can only find lower bounds of entanglement measures, by performing measurements on the correlation [MB07]. In the second part of this thesis, we define and analyze a see-saw method to determine the state and measurement operators that reconstruct both the correlation itself as its entanglement dimension, in a see-saw method of semidefinite programs.

The first part of the thesis provides an improvement over a theoretical result, about how much entanglement one needs to be able to reconstruct all classical correlations. As a new upper bound for the required entanglement dimension, we found that

$$C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma),$$

for $d \geq \dim(C_{\text{loc}})$, which is an improvement over the bound $d \geq \dim(C_{\text{loc}}) + 1$, as given in the literature. This is achieved by demonstrating that the set of correlations using private randomness is connected, which allows for an improved version of Carathéodory's theorem.

In Chapter 2 we start by giving an example explaining the setting of bipartite correlations. Also, we give an overview of several relevant definitions for both classical and quantum correlations. In the classical case, we define the sets $C_{\text{det}}(\Gamma)$ (deterministic correlations), $C_{\text{private}}(\Gamma)$ (private randomness correlations) and $C_{\text{loc}}(\Gamma)$ (shared randomness correlations). To describe the set $C_q^d(\Gamma)$ (quantum correlations), we need the concepts of *Positive Operator Valued Measurements (POVM's)* and an *(entangled) state* . In Chapter 3 we use these definitions to proof the earlier described upper bound for the entanglement dimension such that $C_{\text{loc}}(\Gamma) \in C_q^d(\Gamma)$.

In Chapter 4 we start working towards the reconstruction algorithm, by introducing definitions and results concerning Semidefinite Programming (a particular form of optimization). We also extend the standard SDP program to the complex- and block form-case, to better suit the algorithm we'll use.

In Chapter 5 we describe the see-saw algorithm for the reconstruction of a correlation as a see-saw method of Semidefinite Programs (in complex block form). This is important, as it allows the use of efficient SDP-solvers in the implementation of the algorithm.

Finally, Chapter 6 provides some results of the algorithm, for example for the correlations discussed in Chapter 2. Also, we mention some empirical results from the algorithm, some of which can be explained with the earlier described theory.

This thesis is written as part of the double bachelor's degree in Applied Mathematics and Applied Physics at the Delft University of Technology.

I hope you enjoy reading this work as much as I did creating it!

2

THEORY ON BIPARTITE CORRELATIONS

In this chapter, the basic notation and definitions concerning bipartite correlations are introduced. In Section 2.1, the general setting is described, in which two parties (Alice and Bob) perform experiments in order to answer the questions they receive. In this framework, the concept of a *(bipartite) correlation* will be introduced as a way of describing the relationship between the two parties.

Next, in Section 2.2 we consider how the *answering protocols* of the two parties determine which correlations can be constructed. We start by considering the *classical* answering protocols, in which the use of entanglement is not allowed. While doing this, some important sets of correlations are defined; $C_{\text{det}}(\Gamma)$, $C_{\text{private}}(\Gamma)$, and $C_{\text{loc}}(\Gamma)$.

Finally, the effects of using *entangled states* in the measurements is investigated in Section 2.3. It turns out that the use of entanglement enables a new set of correlations, $C_q(\Gamma)$, which contain correlations that can not be obtained classically. This quantum-mechanical description requires the introduction of some additional notions, such as *quantum states*, a *Positive Operator Valued Measure (POVM)* and the *entanglement dimension*. Some important results are given that connect $C_q(\Gamma)$ with the classically obtainable correlations.

I would like to emphasize that this chapter is mostly a collection of already known definitions and results. For additional resources in which entanglement in the bipartite setting is described, see for example [GLL18], [WW01] or [PV16]. Also note that the use of POVM's and bipartite correlations is only one possible interpretation; alternatively one could express the same ideas in terms of *observables* and *quantum XOR games*, as in [RV15] and [Bri11].

2.1. BIPARTITE CORRELATIONS

2.1.1. GENERAL SETTING

A bipartite correlation is, as the name suggests, a way to describe the relation between two parties. Historically, these parties are called Alice and Bob. They both receive a question from a third party; Alice receives a question s from a "question set" S, and Bob receives a question t from "question set" T. Next, Alice and Bob are required to give an answer to their question; Alice can respond with an answer a from an "answer set" A and Bob can respond with an answer b from his "answer set" B. In this paper, the sets A, B, S and T are assumed to be finite. The set $\Gamma = A \times B \times S \times T$ represents the set of all possible configurations of questions and answers. Crucially, Alice and Bob are not able to communicate after receiving the questions, and they do not know which question the other party received.

We are interested in the probability that Alice and Bob receive the questions (s,t) *and* that they answer with (a,b). The collection of all these probabilities is given by a *bipartite correlation* $P(a,b|s,t) \equiv P$, also known as a *correlation table*, or simply a *correlation*. From this probability interpretation, it is clear that every bipartite correlation must satisfy $P(a,b|s,t) \geq 0$ for all $(a,b,s,t) \in \Gamma$ and $\sum_{a,b} P(a,b|s,t) = 1$ for all $(s,t) \in S \times T$.

Definition 2.1. *Let* $\Gamma = A \times B \times S \times T$. *Then* $P \in \mathbb{R}^{\Gamma}$ *is a* bipartite correlation *if it satisfies :*

1. $P(a,b|s,t) \geq 0$ for all $(a,b,s,t) \in \Gamma$, and
2. $\sum_{a,b} P(a,b|s,t) = 1$ for all $(s,t) \in S \times T$.

Another condition on the bipartite correlations follows from the fact that the parties are unable to communicate after they receive their questions. That is, the probabilities for the answers of Bob should not depend on the answer (or the question) from Alice. These conditions are called the *no-signalling conditions*, which can be expressed as follows:

$$\sum_b P(a,b|s,t) = \sum_b P(a,b|s,t') \qquad \text{for all } a,s,t,t', \tag{2.1}$$

$$\sum_a P(a,b|s,t) = \sum_a P(a,b|s',t) \qquad \text{for all } b,s,t,s'. \tag{2.2}$$

Before we go on with more definitions about bipartite correlations, it is worth improving our understanding of what exactly a bipartite correlation represents. In my opinion, the concept of a bipartite correlation can best be explained by means of an example.

Example 1. Suppose we have the usual setting with Alice and Bob. Since you heard so many great stories about Alice and Bob, you decide to send each of them some of your questions. You would like to ask them about their gender and their year of birth. Obviously, you can't do this, as one must never ask a woman about her age! Therefore, you decide to ask Alice about her name instead. Therefore, the question sets are as follows:

- S = Question set Alice = { Gender , Name }

- T = Question set Bob = { Gender , Birth-year }

Having worked extensively with Alice and Bob over the last couple of months, I can assure you which answers they will give you: Alice identifies as a Woman with the name Alice, and Bob identifies as a Male, born in the year 2000. The answer sets are therefore as follows:

- A = Answer set Alice = { Female , Alice }
- B = Answer set Bob = { Male , 2000 }

We are now ready to think about the *correlation table* $P(a,b|s,t)$, or the probability that Alice and Bob give answers a and b to questions s and t. In this case, the probabilities are trivial, as Alice and Bob will always give the clearly correct answers: the probabilities are either 0 or 1 (this makes the correlation *deterministic*, as we will describe in Section 2.2.1). Therefore, the correlation table looks as follows:

P (Female, Male | Gender, Gender)	1	P (Alice , Male | Gender , Gender)	0
P (Female , Male | Gender , Birth-year)	0	P (Alice , Male | Gender , Birth-year)	0
P (Female , Male | Name , Gender)	0	P (Alice , Male | Name , Gender)	1
P (Female , Male | Name , Birth-year)	0	P (Alice , Male | Name , Birth-year)	0
P (Female, 2000 | Gender, Gender)	0	P (Alice , 2000 | Gender , Gender)	0
P (Female , 2000 | Gender , Birth-year)	1	P (Alice , 2000 | Gender , Birth-year)	0
P (Female , 2000 | Name , Gender)	0	P (Alice , 2000 | Name , Gender)	0
P (Female , 2000 | Name , Birth-year)	0	P (Alice , 2000 | Name , Birth-year)	1

Table 2.1: An example of a bipartite correlation.

Keep in mind that the correlation table $P(a,b|s,t)$ is an element in $\mathbb{R}^{\Gamma}$, so it assigns a real number to every combination of questions and answers. This can be conveniently shown in a table as above. Of course, correlation tables might be more complicated, for example by using probabilities between 0 and 1. This is exactly what we are about to do in the following sections.

In the remainder of this chapter, we consider several types of bipartite correlations. We categorize the correlations based on the resources that are available to Alice and Bob. These resources determine the *answer protocols* that the parties can use. These protocols can, for example, consist of a probabilistic process (randomness), or they can be based on measuring entangled particles. In the next sections, we describe all such protocols. Before introducing the quantum correlations, we start by considering all classical correlations.

2.2. CLASSICAL CORRELATIONS

Classical correlations are all correlations that do not make use of entanglement. The only resource that might be available classically is randomness, either private randomness or shared randomness. In the next three subsections, we consider the three classical answer protocols: the set $C_{\text{det}}(\Gamma)$ of deterministic correlations, the set $C_{\text{private}}(\Gamma)$ of correlations with private randomness and the set $C_{\text{loc}}(\Gamma)$ of correlations with shared randomness. As will become clear, we have that $C_{\text{det}}(\Gamma) \subseteq C_{\text{private}}(\Gamma) \subseteq C_{\text{loc}}(\Gamma)$.

2.2.1. DETERMINISTIC PROTOCOLS

The simplest answer protocol is the *deterministic protocol*. Alice and Bob will decide for every question which answer they will give, without using any randomness. The set of all deterministic correlations is denoted by $C_{\text{det}}(\Gamma)$.

Definition 2.2. *A bipartite correlation is said to be* deterministic *if it can be written as* $P(a,b|s,t) = P_A(a|s)P_B(b|t)$ *for all* $(a,b,s,t) \in \Gamma$, *with:*

1. $P_A(a|s), P_B(b|t) \in \{0,1\}$ for all *(a,b,s,t)* $\in \Gamma$, and
2. $\sum_a P_A(a|s) = \sum_b P_B(b|t) = 1$ for all $(s,t) \in S \times T$.

Example 2. For an example of a deterministic correlation, we refer to Example 1. To see that Definition 2.2 is satisfied, we introduce the personal probabilities $P_A(a|s)$ and $P_B(b|t)$ exactly as one would expect from the story:

P_A (Alice | Name)	1	P_B (Male | Gender)	1
P_A (Alice | Gender)	0	P_B (Male | Birth-year)	0
P_A (Female | Name)	0	P_B (2000 | Gender)	0
P_A (Female | Gender)	1	P_B (2000 | Birth-year)	1

Table 2.2: Personal probabilities $P_A(a|s)$ and $P_B(b|t)$ for deterministic protocol.

It is immediately clear that the conditions in Definition 2.2 are satisfied; all personal probabilities are either 0 or 1, the sum over all answers is one for every question set, and the product of the personal probabilities reproduces the correlation table as in Table 6.1.

2.2.2. PRIVATE RANDOMNESS PROTOCOLS

In the private randomness answer protocol, both parties are able to implement randomness *locally*. That is, they can use probabilistic events to determine their own answer, but they can not both use the same probabilistic event. So instead of using a *public* probabilistic event, Alice and Bob are only allowed to use their own *private* probabilistic event.

The set of all correlations that can be constructed with such private randomness is denoted by $C_{\text{private}}(\Gamma)$. In modern literature, this set is relatively unknown compared to $C_{\text{loc}}(\Gamma)$. In Chapter 3, we show that $C_{\text{private}}(\Gamma)$ is connected.

Definition 2.3. *A bipartite correlation that is based on a* private randomness *protocol can be written as* $P(a,b|s,t) = P_A(a|s)P_B(b|t)$ *for all* $(a,b,s,t) \in \Gamma$, *with:*

1. $P_A(a|s), P_B(b|t) \in [0,1]$ for all *(a,b,s,t)* $\in \Gamma$, and
2. $\sum_a P_A(a|s) = \sum_b P_B(b|t) = 1$ for all $(s,t) \in S \times T$.

Example 3. After receiving insightful answers from Alice and Bob on your somewhat trivial questions, you decide to ask them to perform some experiments for you. You ask Alice and Bob for either the result of flipping a coin, or whether the outcome of throwing a die is even. To make things interesting, you provide Alice with a fair coin and die, but you secretly give Bob an unfair coin and die. The question set is therefore

- S = { Coin, Die }, and
- T = { Coin , Die },

and the answer sets are:

- A = { Heads , Tails , Even , Odd }, and
- B = { Heads, Tails, Even , Odd }.

If Bob's coin gives Heads with probability 0.25 and his die gives an even number with probability 0.10, the personal probabilities will be as follows:

P_A (Heads | Coin)	0.5	P_B (Heads | Coin)	0.25
P_A (Heads | Die)	0	P_B (Heads | Die)	0
P_A (Tails | Coin)	0.5	P_B (Tails | Coin)	0.75
P_A (Tails | Die)	0	P_B (Tails | Die)	0
P_A (Even | Coin)	0	P_B (Even | Coin)	0
P_A (Even | Die)	0.5	P_B (Even | Die)	0.10
P_A (Odd | Coin)	0	P_B (Odd | Coin)	0
P_A (Odd | Die)	0.5	P_B (Odd | Die)	0.9

Table 2.3: Personal probabilities $P_A(a|s)$ and $P_B(b|t)$ for private randomness protocol.

A quick inspection shows that these personal probabilities indeed satisfy Definition 2.3. For example, when we sum over all possible answers when we ask Bob about his Coin, we find $0.25 + 0.75 + 0 + 0 = 1$. With these personal probabilities, the correlation table is now completely defined. Writing it out completely is a bit excessive, considering the fact that $|\Gamma| = 2 \times 2 \times 4 \times 4 = 64$, so $P \in \mathbb{R}^{64}$. As an example, we can see that

$$\mathrm{P}(\text{ Even , Odd} \mid \text{Die , Die }) = P_A\,(\text{ Even} \mid \text{Die }) \cdot P_B\,(\text{ Odd} \mid \text{Die }) = 0.5 \times 0.9 = 0.45.$$

2.2.3. SHARED RANDOMNESS PROTOCOLS

This answer protocol for classical correlations is by far the most important, as it represents all correlations that can be made without entanglement. In a shared randomness protocol, both parties can again use randomness to determine their answer, but this time they can make use of the *same* probabilistic event. That is, they can both observe the same probabilistic event. The set $C_{\text{loc}}(\Gamma)$ of shared randomness correlations is defined as the convex hull of the deterministic correlations.

Definition 2.4. *A bipartite correlation using* shared randomness *can be written as a convex combination of deterministic correlations:*

$$P(a,b|s,t) = \sum_i \lambda_i P_{A,i}(a|s) P_{B,i}(b|t), \tag{2.3}$$

with

1. $\lambda_i \geq 0$ for all i,
2. $\sum_i \lambda_i = 1$,
3. $P_A(a|s), P_B(b|t) \in \{0,1\}$ for all $(a,b,s,t) \in \Gamma$, and
4. $\sum_a P_A(a|s) = \sum_b P_B(b|t) = 1$ for all $(s,t) \in S \times T$.

Example 4. After making Alice and Bob perform so many experiments before, you feel a bit sorry for them. But, since you still have not made all classical correlations, you come up with a new plan. You ask Alice and Bob to watch a live-stream, in which your friend Casino repeatedly throws two (fair) Coins and two (fair) Dice. You use the following questions sets:

- S = { Same_Coins, Same_Dice },
- T = { Number_Heads , Sum_Dice },

on which Alice and Bob may answer with

- A = { Yes, No }, and
- B = { 0 , 1 , 2, . . . , 12 }.

For example, we have that P(Yes , 8 | Same_Coins , Sum_Dice) $= \frac{1}{2} \times \frac{5}{36} = \frac{5}{72}$, but also P(Yes , 7 | Same_Dice , Sum_Dice) = 0. It is clear that this correlation can not be written as a private randomness protocol, since the probabilities are not independent:
0 = P(Yes , 7 | Same_Dice , Sum_Dice) $\neq P_A$(Yes | Same_Dice) $\times P_B$(7 | Sum_Dice) = $\frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$.

Checking if this set-up satisfies Definition 2.4 is not straightforward; let's say I leave it to the reader as an exercise. However, it is certain that this correlation can be written as in 2.3, which we will show in the following subsection.

From this example, it is clear that $C_{\text{private}}(\Gamma) \subseteq C_{\text{loc}}(\Gamma)$, since you can always decide that Alice is only allowed to use half of the live-stream and Bob the other half. In this example, you can ask both Alice and Bob to only use "their own" die and coin from the live-stream.

2.2.4. GENERALIZED CLASSICAL CORRELATIONS

We have now seen all three types of protocols that can be used to construct classical correlations. The major difference between the described sets is the way that randomness is utilized. In general, this randomness can influence the individual answer-probabilities. This can be formalized by introducing a *hidden variable* $\lambda \in \Lambda$, as is done in the following definition.

Definition 2.5. *We say that a bipartite correlation is* classical*, or that it* allows a classical model*, if it can be represented in the following form:*

$$P(a,b|s,t) = \int_{\lambda\in\Lambda} M(d\lambda) P_{A,\lambda}(a|s) P_{B,\lambda}(b|t), \tag{2.4}$$

where M is a probability measure on Λ*, and where for each* λ*,* $P_{A,\lambda}(a|s)$ *and* $P_{B,\lambda}(b|t)$ *are personal probabilities, such that*

1. $P_A(a|s), P_B(b|t) \in \{0,1\}$ for all *(a,b,s,t)* $\in \Gamma$, and
2. $\sum_a P_A(a|s) = \sum_b P_B(b|t) = 1$ for all $(s,t) \in S \times T$.

In this thesis, the sets *A, B, S* and *T* are finite. Therefore, the integral in Equation 2.4 can be written as a convex combination:

$$\begin{aligned} P(a,b|s,t) &= \sum_i \int_{\Lambda_i} M(d\lambda) P_{A,i}(a|s) P_{B,i}(b|t) \\ &= \sum_i \lambda_i P_{A,i}(a|s) P_{B,i}(b|t), \end{aligned} \tag{2.5}$$

where $\lambda_i = \int_{\Lambda_i} M(d\lambda)$ is the probability corresponding to the region Λ_i.

By comparing this definition with 2.3 it is indeed clear that all classical correlations can be written in terms of shared randomness protocols. This makes the set $C_{\text{loc}}(\Gamma)$ the most important set in this section. It has been studied extensively, for example by Bell

[Bel64]. Due to his contributions, the polytope $C_{\text{loc}}(\Gamma)$ is commonly known as the *Bell polytope*, bounded by the *Bell inequalities*. The study of Bell inequalities is still an active research topic [Bru+14] [WW01][PV10].

Note that the terminology of $C_{\text{loc}}(\Gamma)$ is somewhat confusing, as it is not the set of correlations with local randomness. Instead, the "loc" refers to the fact that the correlations can be described by a local hidden variable, λ.

The dimension of $C_{\text{loc}}(\Gamma)$ can be bounded by observing that there are |S||T| conditions of the form $\sum_{a,b} P(a,b|s,t) = 1$ for all $(s,t) \in S \times T$ (from Definition 2.1), so the dimension of $C_{\text{loc}}(\Gamma)$ is at most |Γ|-|S||T|. If we also include other conditions, such as the non-signalling conditions 2.1 and 2.2, it can be shown that the dimension of the Bell polytope is [Wil+08]

$$D = |S|(|A|-1) + |T|(|B|-1) + |S||T|(|A|-1)(|B|-1). \tag{2.6}$$

2.3. BIPARTITE QUANTUM CORRELATIONS

When considering bipartite quantum correlations, it is helpful to imagine that Alice and Bob both receive a particle from a common source, on which they can perform measurements to reach an answer. Moreover, these particles can be entangled, which enables even more correlations.

Before introducing the necessary definitions, it is worth thinking about what kind of correlations can be made, besides the classical correlations from Definition 2.5. In Definition 2.5, the assumption is made that the probabilities $P_{A,\lambda}(a|s)$ and $P_{B,\lambda}(b|t)$ do not depend on the answers of the other party. However, this condition is not required for a bipartite correlation, as demonstrated by Definition 2.1. While the fact that the parties cannot communicate seems to rule out any relationship between the parties, it can actually be achieved by making use of an entangled quantum state.

The quantum mechanical framework of bipartite quantum correlations is commonly modelled in the *tensor model* and in the *commuting model* [Slo16],[GLL18]. For the purposes of this thesis, the description in the tensor model will be sufficient.

In general, a *quantum state* is described by a *density matrix* ρ , which is a $d \times d$ Hermitian positive semidefinite matrix, with trace equal to 1. In this thesis, we often take ρ to be **rank 1**, so that it can be written as $\rho = \psi\psi^*$ with ψ a unit vector $\in \mathbb{C}^d$.

One special feature of quantum mechanics is that a quantum mechanical system can consist of multiple *subsystems* or *parts*, which might be spatially separated. The following definition of entanglement can be interpreted by assuming a tensor product structure on the Hilbert space.

Definition 2.6. *A finite-dimensional pure bipartite state*

$$\psi \in \mathbb{C}^d \otimes \mathbb{C}^d \tag{2.7}$$

is called non-separable if it cannot be written as a simple tensor product $\psi = \psi_1 \otimes \psi_2$.

In this setting, the particles of Alice and Bob are both half of the whole quantum state $\psi \in \mathbb{C}^d \otimes \mathbb{C}^d$. In this way, the measurement performed by Alice can depend on the measurement performed by Bob and vice versa, obtaining a dependency that is impossible

with classical correlations. We often say that the states of the particles are *entangled*, which is equivalent to the states being non-separable.

To make quantum correlations more precise, we model a measurement by introducing a *Positive Operator Valued Measure (POVM)*. (For the definition of a positive semidefinite matrix, see Section 4.1.)

Definition 2.7. *A Positive Operator Valued Measure (POVM) with a possible outcomes is described by a collection of Hermitian positive semidefinite operators $E^1, ..., E^a$ that sum to the identity matrix: $\sum_{i\in[a]} E^i = I$. The probability of observing outcome $i \in [a]$ when measuring pure state ψ is equal to $\langle \psi, E^i\psi\rangle = \mathrm{Tr}(E^i\psi\psi^*)$.*

So if Alice uses the POVM $\{E_s^a\}_{a\in A}$ to answer question $s \in S$ and Bob uses the POVM $\{F_t^b\}_{b\in B}$ to answer question $t \in T$, the probability of observing outcome (a, b) is given by

$$P(a, b|s, t) = \mathrm{Tr}((E_s^a \otimes F_t^b)\psi\psi^*) = \psi^*(E_s^a \otimes F_t^b)\psi. \tag{2.8}$$

If a correlation can be written the form of equation 2.8, it is called a *quantum correlation*. We say it is *realizable in the tensor model in local dimension d* (or in *dimension d^2*) if $\psi \in \mathbb{C}^d \otimes \mathbb{C}^d$ and $E_s^a, F_t^b \in \mathbb{C}^{d\times d}$. The set $C_q^d(\Gamma)$ is used to denote all such correlations, and we define

$$C_q(\Gamma) = \bigcup_{d\in\mathbb{N}} C_q^d(\Gamma). \tag{2.9}$$

Using equation 2.8 for a POVM, we can see that a correlation using a quantum state that is not entangled can also be obtained by a classical correlation. This is demonstrated in the following theorem.

Theorem 2.1. *If a quantum state ψ is not entangled, then the corresponding bipartite correlation is classical. In particular, it can be written as a correlation $P \in C_{private}(\Gamma)$.*

Proof. From Definition 2.6, a non-entangled quantum state ψ can be written as $\psi = \psi_1 \otimes \psi_2$. Substituting this expression in equation 2.8 and using the mixed-product property of the Kronecker product: $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ together with the distributivity of the conjugate transposition: $(A \otimes B)^* = A^* \otimes B^*$, we can write:

$$\begin{aligned} P(a, b|s, t) &= \psi^*(E_s^a \otimes F_t^b)\psi \\ &= (\psi_1 \otimes \psi_2)^*(E_s^a \otimes F_t^b)(\psi_1 \otimes \psi_2) \\ &= (\psi_1^* \otimes \psi_2^*)(E_s^a \otimes F_t^b)(\psi_1 \otimes \psi_2) \\ &= (\psi_1^* E_s^a) \otimes (\psi_2^* F_t^b)(\psi_1 \otimes \psi_2) \\ &= (\psi_1^* E_s^a \psi_1) \otimes (\psi_2^* F_t^b \psi_2) \\ &= (\psi_1^* E_s^a \psi_1)(\psi_2^* F_t^b \psi_2). \end{aligned} \tag{2.10}$$

To demonstrate properties 1. and 2. in definition 2.3, we note that $\sum_a P_A(a|s) = \sum_a(\psi_1^* E_s^a \psi_1) = \psi_1^*(\sum_a E_s^a)\psi_1 = \psi_1^* I \psi_1 = 1$ for all $s \in S$, as ψ is a unit vector. Also, since the POVM E_s^a is positive semidefinite, we have by definition that $\psi_1^* E_s^a \psi_1 \geq 0$, so $\psi_1^* E_s^a \psi_1 \in [0, 1]$. □

It can be shown that an entangled state ψ can always be used to construct a nonclassical correlation P. This raises the question if any classical correlation can be described as a quantum correlation. That is, does $C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma)$? To answer this question, we define the *(minimal) entanglement dimension* as $D_q(P)$, which is the smallest dimension for which $P \in C_q^d(\Gamma)$ in the tensor model .

Definition 2.8. *Let P be a bipartite correlation. We define the (minimal) entanglement dimension of correlation P as*

$$D_q(P) = \min\{d \in \mathbb{N} : P \in C_q^d(\Gamma)\}. \tag{2.11}$$

In words, the entanglement dimension describes the minimal entanglement that is required to construct a given bipartite correlation. This measure for bipartite correlations will be of great interest in the rest of this thesis, as we will work towards a method to determine $D_q(P)$. In the next chapter, we derive a new theoretical upper bound for $D_q(P)$.

2.4. FINAL REMARKS ON CORRELATION RECONSTRUCTION

Before moving, it is important to realize that the examples given in this chapter do not represent the problem we want to solve in this thesis. In the examples, we first described the answering protocols of Alice and Bob, and then derived the correlation table. This is, as we have seen, quite a trivial task. Our goal is, however, to do the reverse process: we want to derive the answering protocols from a given bipartite correlation. In other words, we want to *reconstruct* a given correlation, as well as the POVM's and the state that create it. This is a much harder task, especially when entangled states are involved. Have a look, for example, at the following correlation table (which is now given as an array), in which the meaning of the questions and answers are unknown:

```
[[[[0.24044465753586122, 0.2661454322752907], [0.22690520877331788,
    ↪ 0.2468762754054039]], [[0.22147307310746026,
    ↪ 0.19577229836803078], [0.22778207858411725,
    ↪ 0.20781101195203128]]], [[[0.26475386497278075,
    ↪ 0.2876650922595557], [0.2782933137353241,
    ↪ 0.30693424912944256]], [[0.27332840438389755,
    ↪ 0.25041717709712247], [0.26701939890724047,
    ↪ 0.23837846351312209]]]]
```

Later, in Chapter 6, we come back to this correlation to try to reconstruct it with an algorithm, which we will develop in Chapter 4 and Chapter 5.

3

USING ENTANGLEMENT TO SIMULATE SHARED RANDOMNESS

In the previous chapter, the set $C_{\text{loc}}(\Gamma)$ of bipartite correlations using shared randomness and the set $C_q(\Gamma)$ using a shared quantum state were introduced. In this chapter, the relation between $C_q^d(\Gamma)$ and $C_{\text{loc}}(\Gamma)$ is investigated further. We will show that $C_{\text{loc}}(\Gamma)$ is entirely contained in $C_q^d(\Gamma)$ for a sufficiently large entanglement dimension d. In other words, shared randomness can be simulated by using an entangled state. In this process, we also show that $C_{\text{private}}(\Gamma) \subset C_q^1(\Gamma)$.

The first section consists of a proof that $C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma)$ for $\text{d} \geq \dim(C_{\text{loc}}(\Gamma))$, which is an improvement of the commonly used bound of $\dim(C_{\text{loc}}(\Gamma))$ +1. The reduction is mainly based on the fact that $C_{\text{private}}(\Gamma)$ turns out to be connected, which enables the use of a stronger version of Carathéodory's theorem, without the "+1".

3.1. INTRODUCTION TO THE THEOREM

The fact that $C_q(\Gamma)$ might contain non-classical correlations has been known for a long time. It was first demonstrated in 1976 by Bell [Bel64], who showed that the inclusion $C_{\text{loc}}(\Gamma) \subset C_q(\Gamma)$ is strict whenever A,B,S and T all contain at least two elements.

The proposition in this chapter states something subtly different, as it concerns $C_q^d(\Gamma)$ instead of $C_q(\Gamma)$ (which is the union of $C_q^d(\Gamma)$ over all $d \geq 1$). It is already known that $C_q^d(\Gamma)$ will contain $C_{\text{loc}}(\Gamma)$ if d is sufficiently large. In [Wil+08], the non-signalling conditions 2.1 and 2.2 were used to determine the dimension of $C_{\text{loc}}(\Gamma)$, resulting in Equation 2.6. With Carathéodory's Theorem, this gives the lower bound $\dim(C_{\text{loc}}(\Gamma) + 1$.

In Proposition 3.1, a new lower bound for d is derived:

Proposition 3.1. *For the set $C_{\text{loc}}(\Gamma)$ of correlations with shared randomness and the set $(C_q(\Gamma))$ of correlations using a shared quantum state of dimension d the following inclusion holds:*

$$C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma) \tag{3.1}$$

for $d \geq \dim(C_{\text{loc}}(\Gamma))$.

The most relevant improvement over the current lowerbound is that the "+1" is shown to be unneccesary, as a result of the connectedness of $C_{\text{private}}(\Gamma)$.

Proposition 3.1 will be proved by using the following two lemmas:

Lemma 3.1. *If P is a correlation in* $C_{\text{loc}}(\Gamma)$*, then P can be written as a convex combination of at most N correlations from* $C_{\text{private}}(\Gamma)$*, with* $N = \dim(C_{\text{loc}}(\Gamma))$.

Lemma 3.2. *If a correlation P can be written as a convex combination of N correlations from* $C_{\text{private}}(\Gamma)$*, then* $P \in C_q^N(\Gamma)$.

Together, it is clear that these two lemmas prove Proposition 3.1.

3.2. WRITING P AS A CONVEX SUM OF dim($C_{loc}(\Gamma)$) TERMS

Lemma 3.1. If P is a correlation in $C_{\text{loc}}(\Gamma)$, then P can be written as a convex combination of at most N correlations from $C_{\text{private}}(\Gamma)$, with $N = \dim(C_{\text{loc}}(\Gamma))$.

Proof. The proof of this lemma is strongly based on a result that can be viewed as "Carathéodory's theorem for connected sets". The "normal" Carathéodory's Theorem reads:

Let $S \subset \mathbb{R}^d$ *be a set. Then every point* $x \in$ *conv(S) can be written as a convex combination of at most d+1 points from S.*

However, if S is connected, we have a stronger result [Bun34]:

Let $S \subset \mathbb{R}^d$ *be a connected set. Then every point* $x \in$ *conv(S) can be written as a convex combination of at most d points from S.*

So in order to prove Lemma 3.1, it suffices to show that $C_{\text{private}}(\Gamma)$ is connected, because the theorem above then implies that every P $\in$ conv($C_{\text{private}}(\Gamma)$)=$C_{\text{loc}}(\Gamma)$ can be written as a convex combination of at most dim($C_{\text{loc}}(\Gamma)$) terms.

It remains to be shown that $C_{\text{private}}(\Gamma)$ is indeed connected. This will be done by using the fact that the continuous image of a connected set is connected [Car00].

Suppose we have two arbitrary bipartite correlations $V, W \in C_{\text{private}}(\Gamma)$. From the definition of $C_{\text{private}}(\Gamma)$ in Section 2.2.2, it follows that:

$$\text{there is a } R_0 \in \mathscr{P}(A|S), \text{ and a } Q_0 \in \mathscr{P}(B|T) : V = R_0 \cdot Q_0 \text{ for all } a, b, s, t \in \Gamma, \tag{3.2}$$

and

$$\text{there is a } R_1 \in \mathscr{P}(A|S), \text{ and a } Q_1 \in \mathscr{P}(B|T) : W = R_1 \cdot Q_1 \text{ for all } a, b, s, t \in \Gamma, \tag{3.3}$$

where the set of functions $\mathscr{P}(A|S)$ is defined as the functions P(a|s) in $\mathbb{R}^{|A|x|S|}$ for which:

$$P(a|s) \geq 0 \text{ for all } a, b, \text{ and } \sum_a P(a|s) = 1 \text{ for all } s \in S. \tag{3.4}$$

We will use these two correlations V and W to create a continuous path from V to W, that is entirely in $C_{\text{private}}(\Gamma)$, which shows that $C_{\text{private}}(\Gamma)$ is connected.

We introduce two new correlations for $\lambda \in [0,1]$:

$$C_\lambda(a|s) = (1-\lambda)R_0 + \lambda R_1 \tag{3.5}$$

and

$$D_\lambda(b|t) = (1-\lambda)Q_0 + \lambda Q_1. \tag{3.6}$$

Clearly, $C_\lambda \in \mathscr{P}(A|S)$ and $D_\lambda \in \mathscr{P}(B|T)$ for every $\lambda \in [0,1]$. But now we can define E_λ as:

$$E_\lambda(a,b|s,t) = C_\lambda(a|s) \cdot D_\lambda(b|t), \tag{3.7}$$

so $E_\lambda \in \mathbb{R}^{|A|x|B|x|S|x|T|}$ is also in $C_{\text{private}}(\Gamma)$ for every λ. So E_λ represents a continuous path in $C_{\text{private}}(\Gamma)$ from V to W. Since V and W were chosen arbitrarily, this implies that $C_{\text{private}}(\Gamma)$ is connected. □

3.3. A Convex Sum of N Terms has Entanglement Dimension N

Lemma 3.2. If a correlation P can be written as a convex combination of N correlations from $C_{\text{private}}(\Gamma)$, then P $\in C_q^N(\Gamma)$.

Proof. Suppose there is a correlation $P(a,b|s,t)$ that can be written as a convex combination of N correlations from $C_{\text{private}}(\Gamma)$, that is:

$$P(a,b|s,t) = \sum_{i=1}^{N} \lambda_i \cdot P_{A,i}(a|s) \cdot P_{B,i}(b|t) \tag{3.8a}$$

$$\lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i = 1 \tag{3.8b}$$

$$P_{A,i}(a|s), P_{B,i}(b|t) \in [0,1]. \tag{3.8c}$$

For P to be in $C_q^N(\Gamma)$, we need to show that P can be written in terms of a normalized state ψ and two *POVM's*, as defined in subsection 2.2:

$$P(a,b|s,t) = \text{Tr}((E_s^a \otimes F_t^b)\psi\psi^*). \tag{3.9}$$

We now give a direct construction to determine such a ψ and *POVM's*. We start by defininining the two *POVM's* as diagonal matrices:

$$E_s^a = \text{diag}(P_{A,1}(a|s), P_{A,2}(a|s), ..., P_{A,N}(a|s)), \tag{3.10}$$

$$F_t^b = \text{diag}(P_{B,1}(b|t), P_{B,2}(b|t), ..., P_{B,N}(b|t))). \tag{3.11}$$

We can now write the tensor product $E_s^a \otimes F_t^b$ as follows:

$$E_s^a \otimes F_t^b = \begin{pmatrix} P_{A,1}(a|s) & 0 & \dots & 0 \\ 0 & P_{A,2}(a|s) & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & P_{A,N}(a|s) \end{pmatrix} \otimes \begin{pmatrix} P_{B,1}(b|t) & 0 & \dots & 0 \\ 0 & P_{B,2}(b|t) & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & P_{B,N}(b|t) \end{pmatrix} \tag{3.12}$$

$$= \begin{pmatrix} P_{A,1}(a|s)(F_t^b) & 0 & \dots & 0 \\ 0 & P_{A,2}(a|s)(F_t^b) & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & P_{A,N}(a|s)(F_t^b) \end{pmatrix}. \tag{3.13}$$

So this tensorproduct will again be a diagonal matrix, with on the diagonal the NxN elements

$$P_{A,1}(a|s) \cdot P_{B,1}(b|t), P_{A,1}(a|s) \cdot P_{B,2}(b|t), ..., P_{A,1}(a|s) \cdot P_{B,N}(b|t),$$
$$P_{A,2}(a|s) \cdot P_{B,1}(b|t), P_{A,2}(a|s) \cdot P_{B,2}(b|t), ..., P_{A,N}(a|s) \cdot P_{B,N}(b|t). \tag{3.14}$$

Finally, we define the corresponding state ψ as follows:

$$\psi = \sum_{i=1}^{N} \sqrt{\lambda_i} \cdot \mathrm{e}_i \otimes \mathrm{e}_i, \tag{3.15}$$

where e_i is the i^{th} basisvector with dimension N.

Note that ψ is indeed a unit vector, as

$$||\psi||^2 = \sum_{i=1}^{N} |\sqrt{\lambda_i}|^2 = 1, \tag{3.16}$$

since P was assumed to be a convex sum with weights λ.

Now we get that:

$$\psi\psi^* = (\sum_{i=1}^{N} \sqrt{\lambda_i} \cdot \mathrm{e}_i \otimes \mathrm{e}_i) \times (\sum_{i=1}^{N} \sqrt{\lambda_i} \cdot \mathrm{e}_i \otimes \mathrm{e}_i)^* \tag{3.17}$$

$$= \begin{pmatrix} \lambda_1 & & & \vdots & & \sqrt{\lambda_1\lambda_2} & & \vdots & & & \sqrt{\lambda_1\lambda_N} \\ & 0 & & \vdots & & & & \vdots & & & \\ & & \ddots & \vdots & & & & \vdots & & & \\ \cdots & \cdots & \cdots & \cdot & \cdots & \cdots & \cdots & \cdot & \cdots & \cdots & \cdots \\ & & & \vdots & 0 & & & \vdots & & & \\ \sqrt{\lambda_2\lambda_N} & & & \vdots & & \lambda_2 & & \vdots & & & \sqrt{\lambda_1\lambda_N} \\ & & & \vdots & & & \ddots & \vdots & & & \\ \cdots & \cdots & \cdots & \cdot & \cdots & \cdots & \cdots & \cdot & \cdots & \cdots & \cdots \\ & & & \vdots & & & & \vdots & 0 & & \\ & & & \vdots & & & & \vdots & & \ddots & \\ \sqrt{\lambda_1\lambda_N} & & & \vdots & & \sqrt{\lambda_2\lambda_N} & & \vdots & & & \lambda_N \end{pmatrix}, \tag{3.18}$$

where the dots in the matrix represent NxN blocks within the matrix. On the diagonal, the elements are $a_{i+n-1,i+n-1} = \lambda_i$ for $i = 1, 2, ..., N$.

When multiplying the last two matrices, and using Equation 3.8a, we indeed see that:

$$\mathrm{Tr}((E_s^a \otimes F_t^b)\psi\psi^*) = \sum_{i=1}^{N} \lambda_i P_{A,i}(a|s) P_{B,i}(b|t) = P(a,b|s,t). \tag{3.19}$$

□

Later in the research, while determining the entanglement dimension for a correlation in $C_{\text{private}}(\Gamma)$ (Section 6.1.2), the hypothesis was raised that

$$C_{\text{private}}(\Gamma) \subset C_q^1(\Gamma). \tag{3.20}$$

The proof of this statement follows directly from Lemma 3.2, namely for N=1.

With this result, we end our analysis of the geometry of the sets of bipartite correlations. The result from this section is the most important new contribution to the theory.

In the rest of this thesis, we develop a method to reconstruct a given bipartite correlation, from which we can also derive the entanglement dimension. In the next chapter, we start by summarizing the relevant theory from Semidefinite Programming.

4

THEORY OF SEMIDEFINITE PROGRAMMING

In the previous chapters, the theory of bipartite quantum correlations was discussed, as well as some new theoretical results. In the remainder of this paper, this knowledge is applied in an optimizational context. Specifically, we will investigate how one might reconstruct a given bipartite correlation, by choosing the best possible measurement operators and state. This requires insights form the field of optimization, which are introduced in this chapter.

In section 4.1, we introduce the definition of a *Positive Semidefinite Matrix*, together with some important properties. This type of matrix is essential in Section 4.2, where *Positive Semidefinte Programs* are discussed. Finally, we extend the Semidefinite Programs to the complex case in Section 4.3 and to the block-form case in section 4.4.

Later, in Chapter 5, we use the insights from this chapter to find an explicit construction of measurement operators and state.

4.1. POSITIVE SEMIDEFINITE MATRICES

Semidefinite programming (SDP) is an important topic in optimization, which has been studied extensively in the previous couple of decades. It can be seen as a generalisation of several standard problems, such as linear and quadratic optimization. In semidefinite programming, a linear objective function is minimized, but with nonlinear (but convex!) constraints. Despite the generality of SDP's, research has shown that they can generally be solved in polynomial time, just like linear programming.

For a more extensive survey on semidefinite programming, or on the different methods to find their solution, see for example, [Fre04], [LV12],[VB96].

The crucial aspect of semidefinite programming is that the variable X must lie in the cone of positive semidefinite matrices.

Definition 4.1. *A symmetric $n \times n$ matrix X is called positive semidefinite (PSD) matrix if*

$$v^T X v \geq 0 \qquad \text{for any} \qquad v \in \mathbb{R}^n$$

We use the notation $X \succeq 0$ to denote that X is positive semidefinite. Let S^n denote the set of symmetric $n \times n$ matrices, and let S^n_+ denote the set of positive semidefinite $n \times n$ matrices. We now show that S^n_+ is a closed and convex cone, a property that will be very useful when solving optimizational problems with positive semidefinite matrices. For example, it guarantees that all local minima of an interior point method are equal to the global minimum.

Theorem 4.1. *The set S^n_+ of positive semidefinite $n \times n$ matrices is a closed and convex cone.*

Proof. We first show that S^n_+ is a convex cone, i.e. it is closed under linear combinations with positive coefficients. Let $X, Y \in S^n_+$. Then we have for any scalars $\alpha, \beta \geq 0$ and every $v \in \mathbb{R}^n$ that

$$v^T(\alpha X + \beta Y)v = \alpha v^T X v + \beta v^T Y v \geq 0, \tag{4.1}$$

demonstrating that S^n_+ is a convex cone.

That S^n_+ is closed follows from

$$S^n_+ = \{A \in S^n : x^T A x \geq 0 \forall x \in \mathbb{R}^n\} = \bigcap_{x \in \mathbb{R}^n} \{A \in S^n : x^T A x \geq 0\}. \tag{4.2}$$

We conclude that S^n_+ is an intersection of closed halfspaces, so S^n_+ is closed. Note that this argument is a different way of showing that S^n_+ is convex.

□

For the norm that corresponds to the set $S^n_+ \in \mathbb{R}^{n \times n}$, we first introduce the trace inner product:

Definition 4.2. *For two matrices $X, Y \in \mathbb{R}^{n \times n}$, we define the* trace inner product *as*

$$\langle X, Y \rangle = \mathrm{Tr}(X^T Y) = \sum_{i,j=1}^{n} X_{ij} Y_{ij} \tag{4.3}$$

This defines the Frobenius norm *on $\mathbb{R}^{n \times n}$, obtained by setting $\|X\| = \sqrt{\langle X, X \rangle}$.*

We now give some results that give an insight to the properties of a semidefinite matrix, some of which we will use later on. For a proof of these results, see [LV12].

Theorem 4.2. *Let $X \in S^n$ be a symmetric matrix. The following assertions are equivalent:*

1. *X is positive semidefinite ($X \succeq 0$)*
2. *All eigenvalues of X are non-negative, i.e. the spectral decomposition of X is of the form $X = \sum_{i=1}^n \lambda_i u_i u_i^T$ with all $\lambda_i \geq 0$.*
3. *There exist vectors $v_1, \ldots, v_n \in \mathbb{R}^k$ (for some $k \geq 1$) such that $X_{ij} = v_i^T v_j$ for all $i, j \in [n]$; the vectors v_i are called a Gram representation of X.*
4. *$\langle X, Y \rangle \geq 0$ for all $Y \in S^n_+$*

Also, if $X \succeq 0$ and if $X_{ii} = 0$, then $X_{ij} = X_{ji} = 0$ for all $j = 1, \ldots, n$.

4.2. SEMIDEFINITE PROGRAM

We are now ready to introduce the concept of a semidefinite program. Generally speaking, we want to optimize the variable matrix X for a linear objective function, characterized by the matrix C. The objective function can be compactly written as $\langle C, X\rangle$, using the trace inner product from 4.2. Since X is symmetric (because it is PSD), we can also assume, without loss of generality, that the objective matrix C is symmetric as well. The variable X can either be viewed as a symmetric $n \times n$ matrix, as an array of n^2 components $(x_{11}, x_{12}, \ldots, x_{nn})$, or even as a vector in the space S^n.

There are two types of constraints in an SDP. The first one is that the variable matrix X must be a positive semidefinite matrix: $X \succeq 0$. It might be useful to draw the analogy with an LP, by thinking of the constraint $X \succeq 0$ as stating that all *n eigenvalues* of X must be nonnegative, in contrast to an LP where all *n components* of x must be nonnegative.

The second constraint consists of m linear equations that X must satisfy. All m constraints can be written by using m symmetric matrices $A_1, \ldots, A_m$, and the vector $b \in \mathbb{R}^m$: $\langle A_i, X\rangle = b_i$ for $i \in [m]$.

Definition 4.3. *Let $C \in S^n$, $A_i \in S^n$ and $b_i \in \mathbb{R}^n$ for $i \in [m]$ be given, and let $X \in S^n$ be a matrix variable. A semidefinite program (SDP) is an optimization problem of the form:*

$$\begin{aligned} \min \quad & \langle C, X\rangle && && (4.4a)\\ s.t. \quad & \langle A_i, X\rangle = b_i && \text{for } i \in [m] && (4.4b)\\ & X \succeq 0. && && (4.4c) \end{aligned}$$

To illustrate the resemblance with an LP, we give a short example for $n = 3$ and $m = 2$. Suppose the following matrices are given:

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 4 & 0 & 3 \\ 0 & 2 & 1 \\ 3 & 1 & 5 \end{pmatrix}, \quad \text{and} \quad C = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \\ 4 & 7 & 2 \end{pmatrix},$$

with $b_1 = 12$ and $b_2 = 20$. Note that all matrices are symmetric, as required. The matrix X will be a 3×3 symmetric matrix: $X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33}. \end{pmatrix}$

Using the symmetry of X, we can rewrite the objective function 4.5a as:

$$\begin{aligned} \langle C, X\rangle &= 1x_{11} + 2x_{12} + 4x_{13} + 2x_{21} + 5x_{22} + 7x_{23} + 4x_{31} + 7x_{32} + 2x_{33} \\ &= 1x_{11} + 4x_{12} + 8x_{13} + 5x_{22} + 14x_{23} + 2x_{33}. \end{aligned}$$

The complete SDP can therefore be written as:

$$
\begin{aligned}
\min \quad & 1x_{11} + 4x_{12} + 8x_{13} + 5x_{22} + 14x_{23} + 2x_{33} \\
\text{s.t.} \quad & 1x_{11} + 4x_{12} + 6x_{13} + 4x_{22} + 10x_{23} + 6x_{33} = 12 \\
& 8x_{11} + 0x_{12} + 6x_{13} + 2x_{22} + 2x_{23} + 5x_{33} = 20 \\
& \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \succeq 0.
\end{aligned}
$$

4.3. COMPLEX SEMIDEFINITE PROGRAM

The description of a semidefinite program in Definition 4.3 is not sufficient in the context of bipartite correlations, as the POVM's and the density matrix might be complex. Therefore, we need to extend the notion of a semidefinite program to complex matrices. Such a program is called a complex (or Hermitian) semidefinite program.

In a complex SDP, the $n \times n$ matrices $A_1, \ldots, A_m, C$ belong to H^n, the set of Hermitian $n \times n$ matrices. Furthermore, the variable X is no longer symmetric, but $X \in H^n_+$, the set of $n \times n$ Hermitian PSD matrices.

Note that the trace inner product will now contain a Hermitian transpose: $\langle X, Y \rangle = \mathrm{Tr}(X^* Y) = \sum_{i,j=1}^{n} \overline{X_{ij}} Y_{ij}$

Definition 4.4. *Let $C \in H^n$, $A_i \in H^n$ and $b_i \in \mathbb{R}^n$ for $i \in [m]$ be given, and let $X \in H^n$ be a matrix variable. A complex semidefinite program is an optimization problem of the form:*

$$
\begin{aligned}
\min \quad & \langle C, X \rangle && && \text{(4.5a)} \\
\textit{s.t.} \quad & \langle A_i, X \rangle = b_i && \text{for } i \in [m] && \text{(4.5b)} \\
& X \succeq 0. && && \text{(4.5c)}
\end{aligned}
$$

The vector b_i is always a real vector. This result is proven in the following theorem.

Theorem 4.3. *For all $A, B \in H^n$, the complex trace inner product $\langle A, B \rangle$ is real.*

Proof. Let $A, B \in H^n$. Since A is Hermitian, we have $A^T = \overline{A}$. Using some basic properties of the trace, we get:

$$\langle A, B \rangle = \mathrm{Tr}(A^* B) = \mathrm{Tr}((A^* B)^T) = \mathrm{Tr}(B^T (A^*)^T) = \mathrm{Tr}(B^T \overline{A}) = \mathrm{Tr}(\overline{A} B^T)$$

and

$$\overline{\langle A, B \rangle} = \overline{\mathrm{Tr}(A^* B)} = \mathrm{Tr}(\overline{A^* B}) = \mathrm{Tr}(\overline{A^*}\ \overline{B}) = \mathrm{Tr}(A^T \overline{B}) = \mathrm{Tr}(\overline{A} B^T)$$

We see that $\langle A, B \rangle = \overline{\langle A, B \rangle}$, so we conclude that $\langle A, B \rangle \in \mathbb{R}$. □

4.4. SEMIDEFINITE PROGRAM IN BLOCK-FORM

As the last step before formulating an SDP for computing bipartite correlations, we need to consider how the program can deal with multiple variables and constraints specifically. In particular, how can we formulate an SDP that minimizes an objective function over multiple positive semidefinite matrices $X^1,\dots,X^K$?

Luckily, it turns out that the most straightforward way of dealing with this problem is also efficient from an algorithmical viewpoint. All matrix variables $X^1,\dots,X^K$ and coefficient matrices $C^1,\dots,C^K$ can be compactly written as a single matrix in block-form, resembling the notation from Definition 4.4:

$$X = \begin{pmatrix} X^1 & & 0 \\ & \ddots & \\ 0 & & X^K \end{pmatrix}, \tag{4.6}$$

and

$$C = \begin{pmatrix} C^1 & & 0 \\ & \ddots & \\ 0 & & C^K \end{pmatrix}. \tag{4.7}$$

Similarly, the $m \times K$ constraint matrices can be written as m matrices $A_1,\dots,A_m$, where every matrix A_i is a block-form representation of the matrices $A_i^1,\dots,A_i^K$:

$$A_i = \begin{pmatrix} A_i^1 & & 0 \\ & \ddots & \\ 0 & & A_i^K \end{pmatrix}. \tag{4.8}$$

In order to use these matrices in an SDP, it is required that that a block-form matrix consisting of PSD matrices is itself PSD. This is demonstrated in the following Theorem.

Theorem 4.4. *Let X be a block-form matrix, containing the positive semidefinite matrices $X^1,\dots,X^K$, as shown above. Then X is also positive semidefinite.*

Proof. Let X be an arbitrary block-form matrix, containing the positive semidefinite matrices $X^1,\dots,X^K$. By the definition of an eigenvalue, every eigenvalue λ of X satisfies

$$\det\begin{pmatrix} X^1-\lambda I & & 0 \\ & \ddots & \\ 0 & & X^K-\lambda I \end{pmatrix} = 0.$$

Rewriting this determinant, we find

$$\det\begin{pmatrix} X^1-\lambda I & & 0 \\ & \ddots & \\ 0 & & X^K-\lambda I \end{pmatrix} = \prod_{k=1}^{K} \det(X^k-\lambda I) = 0.$$

So the eigenvalues of X are precisely the eigenvalues of $X^1,\dots,X^K$. Since those are all nonnegative, we can conclude that X is positive semidefinite.

□

Now that we know that a block-form of PSD matrices is itself PSD, we are ready to define a (complex) semidefinite program in block form. The notation is somewhat cumbersome, but keep in mind how the matrices X, C and $A_1, \ldots, A_m$ are defined in equations 4.6,4.7,4.8.

Definition 4.5. *Let the Hermitian matrices X, C and $A_1, \ldots, A_m$ be given as described above. We denote the dimensions of the k^{th} matrix on the diagonal as $n_k \times n_k$. The dimensions of the matrices X, C and $A_1, \ldots, A_m$ are equal to $\sum_{k=1}^{K} n_k = n$. Also, let $b_i \in \mathbb{R}^n$ for $i \in [m]$ be given. A complex semidefinite program in block-form is an optimization problem of the form:*

$$\min \quad \sum_{k=1}^{K} \langle C^k, X^k \rangle \tag{4.9a}$$

$$s.t. \quad \sum_{k=1}^{K} \langle A_i^k, X^k \rangle = b_i \qquad \text{for } i = 1, \ldots, m \tag{4.9b}$$

$$X^k \in H_+^{n_k} \qquad \text{for } k = 1, \ldots, K. \tag{4.9c}$$

Although the sub- and superscripts might seem somewhat confusing, it is actually very convenient that we can write an SDP in block-form. In practice, it turns out that the efficiency of an SDP solver depends on the size of the largest matrix in such a block-form. Therefore, the use of block-form notation is also efficient from an algorithmic point of view.

In the next chapter, we demonstrate that the problem of reconstructing a bipartite quantum correlation can be solved by using an algorithm based on consecutive Semidefinite Programs.

5

AN SDP FOR THE RECONSTRUCTION OF BIPARTITE QUANTUM CORRELATIONS

With the theoretical background provided in the previous chapter, we are now ready to formalize our problem as a Semidefinite Program, although this still requires some more work.

In Section 5.1, we consider the problem of reconstructing a bipartite quantum correlation in more detail. Also, we describe an algorithm that is able to reconstruct the correlation, based on a see-saw method of Semidefinite Programs. This is a new way of solving this problem.

Next, in Sections 5.2 and 5.3, we demonstrate that the algorithm consists of repeated applications of Semidefinite Programs. We start by giving some crude descriptions of the problem, and then gradually improve our notation, working towards the definition of a complex block-form SDP as in Equation 4.9.

5.1. SEE-SAW ALGORITHM

Loosely speaking, we want to approximate a given bipartite correlation $P(a,b|s,t)$ with an approximation $\widetilde{P}(a,b|s,t)$. As discussed in Section 2.3, a general bipartite correlation can be written as

$$\widetilde{P}(a,b|s,t) = \mathrm{Tr}((\widetilde{E_s^a} \otimes \widetilde{F_t^b})\widetilde{\psi}\widetilde{\psi}^*), \tag{5.1}$$

where a tilde indicates the approximation instead of the real value, which is generally unknown. Before we go into the correct notation for this problem, we first give a general remark about the method of solving this problem.

A close inspection of Equation 5.1 gives that there are three quantities over which can be optimized: the two POVM's E_s^a and F_t^b, and the quantum state $\widetilde{\psi}\widetilde{\psi}^*$, which could also

be written as a density matrix $\widetilde{\rho}$. Instead of immediately solving for all three quantities, we solve for them one after another. This is useful, as we will see that this enables the use of consecutive semidefinite programs, which can be solved efficiently. The idea is that we use two approximations to determine a better approximation for the third quantity. This method is known as the see-saw method, the inspiration for which comes from [PV10] and [IIA06]. In these applications, the see-saw method is used to optimize a linear function, for example a Bell-inequality. In this thesis we use the method with a different goal, namely to approximate a correlation instead of a linear function.

The method is briefly summarized in the following algorithm.

Algorithm 1: Determining minimal entanglement dimension

Data: Bipartite correlation $P(a,b|s,t) \in \mathbb{R}^{\Gamma}$,
number of randomly generated samples for one entanglement dimension *: max_samples*,
precision $\epsilon \in \mathbb{R}$,
maximum number of iterations of see-saw algorithm *max_iterations*,
maximum entanglement dimension *d_max*
Result: Minimal entanglement dimension $D_q(P)$, with corresponding approximations $\widetilde{E_s^a}, \widetilde{F_t^b}$ and $\widetilde{\psi}$
initialization: $d = 1$
while $d <$ *d_max* **do**
  for *i in [1:max_samples]* **do**
    generate random $\widetilde{F_t^b}$ and $\widetilde{\rho}$ of dimension d
    for iterations in [1:max_iterations] **do**
      Solve SDP for $\widetilde{E_s^a}$;
      Update $\widetilde{E_s^a}$;
      Solve SDP for $\widetilde{F_t^b}$;
      Update $\widetilde{F_t^b}$;
      Solve SDP for $\widetilde{\rho}$;
      Update $\widetilde{\rho}$;
      if $\max_{a,b,s,t} |\widetilde{P}(a,b|s,t) - P(a,b|s,t)| \leq \epsilon$ **then**
        stop and return d
      end
    end
  end
  $d += 1$
end

The precise implementation of this implementation is irrelevant for the purposes of this chapter, although the code does contain some interesting ideas, for example how to generate the random POVM's, or how to solve the SDP's efficiently. For the code, we refer to Appendix A.

In the remainder of this chapter, we are only interested in demonstrating that the problems

- Solve for $\widetilde{E_s^a}$, given $P(a,b|s,t)$, $\widetilde{F_t^b}$ and $\widetilde{\rho}$
- Solve for $\widetilde{F_t^b}$, given $P(a,b|s,t)$, $\widetilde{E_s^a}$ and $\widetilde{\rho}$
- Solve for $\widetilde{\rho}$, given $P(a,b|s,t)$, $\widetilde{E_s^a}$ and $\widetilde{F_t^b}$

are indeed Semidefinite Programs. The notation we introduce to show this is not necessary for the implementation, since the solvers can work with the constraints only. However, it is important to know for sure that the problems are SDP, because that allows the use of efficient SDP-solvers.

5.2. SDP FOR POVM'S

In the problem that approximates the best possible POVM's E_s^a, we assume that F_t^b and ρ are given.

As a first try, we could describe our problem as follows:

$$\begin{aligned} \min \quad & \max_{a,b,s,t} |\widetilde{P}(a,b|s,t) - P(a,b|s,t)| && \text{(5.2a)} \\ \text{s.t.} \quad & \ldots, && \text{(5.2b)} \end{aligned}$$

which minimizes the absolute distance between the estimated probability and the real probability for every possible combination of questions and answers. We here use the maximum norm, although other norms could be chosen as well.

One problem with this notation is that the objective function is not linear, so the problem is not SDP. The solution is to encorporate the current objective function in the constraints:

$$\begin{aligned} \min \quad & \mathrm{m} && \text{(5.3a)} \\ \text{s.t.} \quad & \widetilde{P}(a,b|s,t) - P(a,b|s,t) \leq \mathrm{m} \text{ for all } (a,b,s,t) \in \Gamma && \text{(5.3b)} \\ & \widetilde{P}(a,b|s,t) - P(a,b|s,t) \geq -\mathrm{m} \text{ for all } (a,b,s,t) \in \Gamma && \text{(5.3c)} \\ & \ldots. && \text{(5.3d)} \end{aligned}$$

Now the objective function is clearly linear, but the constraints contain inequalities, whereas equalities are required in Equation 4.9. As is commonly the case in such problems, the use of slack variables solves this problem. Because every inequality has its own slack variable, we use the matrices S_1 and S_2, which are $\Gamma \times \Gamma$ diagonal matrices consist-

ing of $s_1(a,b|s,t)$ and $s_2(a,b|s,t)$ respectively:

$$\begin{aligned}
&\min && \mathrm{m} && (5.4a)\\
&\text{s.t.} && \widetilde{P}(a,b|s,t) - P(a,b|s,t) = \mathrm{m} - s_1(a,b|s,t) \text{ for all } (a,b,s,t) \in \Gamma && (5.4b)\\
& && \widetilde{P}(a,b|s,t) - P(a,b|s,t) = -\mathrm{m} + s_2(a,b|s,t) \text{ for all } (a,b,s,t) \in \Gamma && (5.4c)\\
& && S_1, S_2 \in H_+^\Gamma && (5.4d)\\
& && \ldots. && (5.4e)
\end{aligned}$$

This last SDP is ready to be written in the form of 4.9, so in terms of matrices. Keep in mind that this description is for solving $\widetilde{E_s^a}$ for given $\widetilde{F_t^b}$ and $\widetilde{\rho}$. The method for solving $\widetilde{F_t^b}$ is of course analogous.

In our problem, the matrices $C = \mathrm{diag}(C_1, \ldots, C_k)$ and $X = \mathrm{diag}(X_1, \ldots, X_k)$ are of the form:

$$X = \begin{pmatrix} m & & & & & & \\ & S_1 & & & & & \\ & & S_2 & & & & \\ & & & E_1^1 & & & \\ & & & & \ddots & & \\ & & & & & E_s^a \end{pmatrix}, \tag{5.5}$$

and

$$C = \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}. \tag{5.6}$$

In particular, we see that $X^1 = \mathrm{m}$, $X^2 = S_1$, $X^3 = S_2$, $X^4 = E_1^1$, et cetera. Also, $C^1 = 1$, $C^2 = C^3 = \cdots = C^K = 0$. The fact that X and K are PSD follows from Theorem 4.4.

The way that the constraints are written in Equations 5.4b and 5.4c do not immediately seem to be of the form of Equation 4.9b. We will not attempt to do this, as the explicit form of the matrices A_i^k are not interesting to us. It is certain however, that such matrices must exist, because the map $E_s^a \mapsto \mathrm{Tr}((E_s^a \otimes F_t^b)\rho)$ is linear. And if $f(X) \mapsto \mathbb{R}$ is linear, then there is a matrix Y such that $f(X) = \langle X, Y \rangle$. This guarantees that Equation 4.9b is satisfied. We are not interested in the exact form of the matrices A_i^k, as the software used for solving an SDP will also work with the constraints in 5.4b and 5.4c.

We now proceed to the remaining constraints in Equation 5.4e. One property of a POVM is that the sum over all answers must give the identity matrix (see Definition 2.7). This can be written in the form of 4.9 as follows:

$$\sum_a \langle E_s^a, \mathbb{1}_{ij} \rangle = \delta_{ij} \text{ for all } s, i, j, \tag{5.7}$$

where $\mathbb{1}_{ij}$ is a matrix with only zeroes, except a 1 on (i,j).

This completes the demonstration that the problem to approximate E_s^a is a PSD of the form from Equation 4.9.

5.3. SDP FOR DENSITY MATRIX

In the problem that approximates a density matrix, we assume that both POVM's E_s^a and F_t^b are given. The argument that this is an SDP is very similar to the argument from the previous section. One difference is that the matrix X in the objective function now becomes (compare with Equation 5.5):

$$X = \begin{pmatrix} m & & & \\ & S_1 & & \\ & & S_2 & \\ & & & \rho \end{pmatrix}. \tag{5.8}$$

The eigenvalues of the density matrix ρ are probabilities, so they are nonnegative. Therefore, ρ is PSD and so is X.

The other difference is that the constraint 5.7 is replaced by the constraint

$$\mathrm{Tr}(\rho) = \langle I, \rho \rangle = 1. \tag{5.9}$$

It is now clear that the see-saw algorithm 1 consists of repeated Semidefinite Programs. This allows the use of efficient SDP-solvers. Although a single SDP problem can be solved in polynomial time, the problem as a whole is still NP-hard in the number of states [Sta18].

In the next chapter, the results of the algorithm are demonstrated for different types of correlations. With the demonstration in this section, we can be sure that we are allowed to use efficient SDP solvers in that implementation.

6

RESULTS OF THE ALGORITHM

In this final chapter, we give the result of several numerical experiments to demonstrate that the algorithm described in the previous section works in practice. This qualifies more as a proof of concept than as an extensive analysis. The code to generate the results from this chapter can be found in the Appendix, so that you could also test some correlations yourself.

In Section 6.1, we start by testing the algorithm for its intended use: the reconstruction of a given correlation. We see that the algorithm is able to successfully reproduce the classical correlations from $C_{\text{det}}(\Gamma)$, $C_{\text{private}}(\Gamma)$ and $C_{\text{loc}}(\Gamma)$, by using the correlations from the examples in Chapter 2. We also discuss the limitations of the algorithm, which seem to be mainly caused by computational limitations. Also, the numerical experiments demonstrate several interesting phenomena, which motivate us to formulate some new results. For example, we find that every correlation with |A|=1 or |B|=1 has entanglement dimension 1, so that the result from Chapter 3 implies that it can be written as a private randomness correlation.

In Section 6.2 we analyze the behaviour of the algorithm by running it many times with randomly generated correlations. We'll see that all randomly generated quantum correlations can be reconstructed within the expected dimension, but sometimes even in a lower dimension. Analyzing the probabilities of those instances with a lower entanglement dimension than the created dimension can give an insight in the relative volumes of $C_q^d(\Gamma)$ for different dimensions. Finally, we propose a method to generate a random correlation with a given entanglement dimension.

6.1. ANALYSIS OF GIVEN CORRELATIONS

In this section, we test the results of the algorithm by comparing them to theoretically known results. Also, we give some empirical results based on the behaviour of the algorithm.

6.1.1. GIVEN DETERMINISTIC CORRELATIONS

We start with the simplest case: is the algorithm able to reconstruct a given deterministic correlation? To check this, we go back to Example 1, for which we found the following correlation table:

P (Female, Male | Gender, Gender)	1	P (Alice , Male | Gender , Gender)	0
P (Female , Male | Gender , Birth-year)	0	P (Alice , Male | Gender , Birth-year)	0
P (Female , Male | Name , Gender)	0	P (Alice , Male | Name , Gender)	1
P (Female , Male | Name , Birth-year)	0	P (Alice , Male | Name , Birth-year)	0
P (Female, 2000 | Gender, Gender)	0	P (Alice , 2000 | Gender , Gender)	0
P (Female , 2000 | Gender , Birth-year)	1	P (Alice , 2000 | Gender , Birth-year)	0
P (Female , 2000 | Name , Gender)	0	P (Alice , 2000 | Name , Gender)	0
P (Female , 2000 | Name , Birth-year)	0	P (Alice , 2000 | Name , Birth-year)	1

Table 6.1: The correlation table from Example 1

We use the following function and parameters to run the code:

```
A=2
B=2
S=2
T=2
num_iterations = 20
samples = 5
d=1
d_max = 4
epsilon = 10^-8
P_det = [
[
[[0,0],[0,0]],
[[0,0],[0,0]]
],
[
[[0,0],[0,0]],
[[0,0],[0,0]]
]
]
P_det[1][1][1][1]=1
P_det[1][2][1][2]=1
P_det[2][1][2][1]=1
P_det[2][2][2][2]=1

P_used = P_det
println(givenP_increasing_d(P_used,num_iterations,d_max,epsilon,A,B,S
    ↪ ,T))
```

The algorithm is consistently able to reproduce the correlation in dimension 1, within the first sample. The error progression shows that the approximation improves dramatically in the second SDP, in which the second POVM is optimized. A typical progression of the errors is:

```
[0.7461100469238708, 0.3936224345080166, 9.36217736668965e-9],
```

where the first error is after no optimization, the second error after optimizing E and the third error after optimizing F. The recovered POVM's are of the form:

- $E_1^1 = E_2^2 = 1$ and $E_1^2 = E_2^1 = 0$
- $F_1^1 = F_2^2 = 1$ and $F_1^2 = F_2^1 = 0$,

and the recovered state is of course $\rho = 1$; the only state with dimension 1 that has a trace of 1.

It can in fact easily be shown that every deterministic correlation is in $C_q^1(\Gamma)$, as is suggested by this result.

6.1.2. GIVEN PRIVATE RANDOMNESS CORRELATIONS

We now perform the same procedure for the correlations of private randomness. As an example, we test if the algorithm can recover the correlation from Example 3. We use the following function and parameters to run the code (inserting the correlation is somewhat cumbersome, as $P \in \mathbb{R}^{64}$):

```
A=4
B=4
S=2
T=2
num_iterations = 10
samples = 2
d_max = 4
epsilon = 10^-8

function P_A_priv(a,s)
    if a==1 && s==1
        return 0.5
    elseif a==2 && s==1
        return 0.5
    elseif a==3 && s==2
        return 0.5
    elseif a==4 && s==2
        return 0.5
    else
        return 0
    end
end
```

```
function P_B_priv(b,t)
    if b==1 && t==1
        return 0.25
    elseif b==2 && t==1
        return 0.75
    elseif b==3 && t==2
        return 0.1
    elseif b==4 && t==2
        return 0.9
    else
        return 0
    end
end


P_pcorr = [[[[P_A_priv(a,s)*P_B_priv(b,t) for t=1:T] for s=1:S] for b
    ↪ =1:B] for a=1:A]

println("Start")
println("====================================")
println(givenP_increasing_d(P_pcorr,num_iterations,d_max,epsilon,A,B,
    ↪ S,T))
```

Again, the algorithm easily reproduces the given correlation, with only one progression of the see-saw algorithm. The entanglement dimension is 1, and the recovered POVM's found are equal to the personal probabilities:

- $E_s^a = P(a|s)$ for all a, s, and
- $F_t^b = P(b|t)$ for all b, t.

This result hints at the fact that $C_{\text{private}}(\Gamma) \subset C_q^1(\Gamma)$, which indeed follows immediately from Lemma 3.2.

6.1.3. GIVEN SHARED RANDOMNESS CORRELATIONS

We now perform the same procedure for the correlations with shared randomness. Except... the correlation we used in Example 4 is a bit too complicated to implement in the code. Of course, we could approximate the correlation by running simulations, but that is bound to give problems for the entanglement dimension, as the algorithm has a high precision (usually epsilon=10^{-8}). How the algorithm responds to such small perturbations is an interesting topic for future research.

For the purpose of this section, we simplify Example 4 a bit, by using the following question sets:

- S = { Same_Coins, Same_Dice },

- T = { Number_Heads , Number_Ones },

so that the answer sets are

- A = { Yes, No }, and
- B = { 0 , 1 , 2 }.

The following code runs the algorithm for this instance of the problem:

```
A=2
B=2
S=2
T=2
num_iterations = 100
samples = 5
d_max = S*(A-1)+T*(B-1)+S*T*(A-1)*(B-1)
epsilon = 10^-8

P_shar = [[[[0.0 for t=1:T] for s=1:S] for b=1:B] for a=1:A]

P_shar[1][1][1][1]=0.25
P_shar[1][1][1][2]=0.5*5/6*5/6
P_shar[1][1][2][1]=1/4*1/6
P_shar[1][1][2][2]=5/36
P_shar[1][2][1][1]=1/4
P_shar[1][2][1][2]=0.5*2*1/6*5/6 + 0.5*1/36
P_shar[1][2][2][1]=0.5*1/6 + 0.25*1/6
P_shar[1][2][2][2]=1/36
P_shar[2][1][1][1]=0.0
P_shar[2][1][1][2]=0.5*5/6*5/6
P_shar[2][1][2][1]=1/4*5/6
P_shar[2][1][2][2]=5/6*4/6
P_shar[2][2][1][1]=1/2
P_shar[2][2][1][2]=0.5*2*1/6*5/6 + 0.5*1/36
P_shar[2][2][2][1]=0.5*5/6+5/6*1/4
P_shar[2][2][2][2]=2*1/6*5/6

println("Start")
println("====================================")
println(givenP_increasing_d(P_shar,num_iterations,d_max,epsilon,A,B,S
    ↪ ,T))
```

Note that the parameters *samples* and *num_iterations* are set at larger values, since we expect to find a larger entanglement dimension.

This time, the algorithm is not able to reproduce the correlation using entanglement dimension 1. For example, the example correlation at the end of Chapter 2 has entanglement dimension 2. Interestingly, the outcome of the algorithm is not deterministic

anymore; the code from the example above usually finds a construction in either dimension 4 or dimension 5, but sometimes even dimension 6. This suggests that the use of multiple samples is indeed necessary to determine the entanglement dimension accurately. Furthermore, calculations at such high dimensions might cause problems due to machine precision. This could be prevented by using higher order solvers.

During the numerical experiments, the hypothesis arose that all correlations with |A|=1 or |B|=1 have entanglement dimension 1. This result is easily proven, as it follows immediately from the dimension of $C_{\mathrm{loc}(\Gamma)}$ in combination with our improved result from Chapter 3. Note that this proof was not valid with the old result!

6.2. RANDOMLY GENERATED CORRELATIONS

6.2.1. RECONSTRUCTING QUANTUM CORRELATIONS

We now check if the algorithm is also able to reconstruct a quantum correlation. To this end, we first define a method to create a quantum correlation in dimension d. Next, we "forget" the POVM's and state that we used to create the correlation, and try to approximate it with the algorithm. The implementation is as follows:

```
function randomP_increasing_d(num_iterations,d,d_max,epsilon,A,B,S,T)
    #The complete algorithm, determining the correlation and the
        ↪ entanglement
    # dimension
    # This version generates a P itself
    # In most cases, a solution is found for samples =1. However,
        ↪ there have
    # been instances where samples was higher.
    # There is a bug, in which the POVM's vanish, causing an error
    rho, E, F = generaterandomoperators(d,A,B,S,T)
    P_real = generateP(rho, E, F,d,A,B,S,T) # We want to approximate
        ↪ this P
    d=1
    errors = zeros(0)
    while d<d_max+1
        for sa in 1:samples
            errors = zeros(0)
            rho, E, F = generaterandomoperators(d,A,B,S,T) # We
                ↪ initialize with new rho,E,F
            for i in 1:num_iterations
                    myerror = maximum(norm(P_real[a][b][s][t] -
                        ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                        ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                    push!(errors,myerror)
                    println(errors)
                E = findE(F,rho,P_real,d,A,B,S,T)
                    myerror = maximum(norm(P_real[a][b][s][t] -
                        ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
```

```
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
                println(errors)
            F = findF(E,rho,P_real,d,A,B,S,T)
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
                println(errors)
            rho = findrho(E,F,P_real,d,A,B,S,T)
            if myerror < epsilon
                return errors, d,sa,E,F,"Success"
            end
        end
    end
    d+=1
  end
  return errors,"Failed"
end

A=3
B=2
S=2
T=4
d=4
num_iterations = 20
samples = 2
d_max = d+1
epsilon = 10^-8

println("Start")
println("====================================")
println(randomP_increasing_d(num_iterations,d,d_max,epsilon,A,B,S,T))
```

The results of the algorithm are very interesting. In the first place, with this method the algorithm was able to correctly reconstruct all correlations thus far. However, sometimes the algorithm gets stuck in a local optimum, justifying the use of multiple samples per dimension.

Interestingly, the entanglement dimension that the algorithm returns is not always equal to the dimension that was used to create the random correlation; it might also be lower (but never higher). This is to be expected, as we now that $C_q^i(\Gamma) \subset C_q^(\Gamma)$ for $i < j$. In some cases, a randomly generated correlation in a high dimension turns out to be a classical correlation in $C_q^1(\Gamma)$. By analyzing the algorithm over many runs, information can be deduced about the "relative volume" of the quantum sets. This interesting behaviour would be a nice topic of future research.

Finally, one might be interested in creating a correlation with a particular entangle-

ment dimension. The method of randomly creating a correlation in that dimension does not always work, as we just saw that the entanglement dimension is often lower than the dimension in which the correlation is created. As a possible solution, one might try to first solve an optimization problem, in which a linear objective function is minimized over the set of quantum correlation in dimension d. This results in a correlation, which can be reconstructed using the methods from this thesis.We propose the hypothesis that such a correlation is more likely to have entanglement dimension d compared to a randomly generated correlation in dimension d. However, future research is necessary to confirm this hypothesis.

7

CONCLUSION

In the first chapter of this thesis, a thorough mathematical description of bipartite quantum correlations is made. This description considers quantum correlations by using POVM's and the tensor model. Also, an effort has been made to explain the theory in a simple manner, by introducing multiple examples.

This theory is used to analyze the geometry of several sets of bipartite correlations. In particular, it is investigated how much entanglement is necessary to reconstruct all classical correlations. This analysis has led to an improved upper bound on the entanglement dimension for which $C_{\text{loc}}(\Gamma) \subseteq C_q^d(\Gamma)$, namely for $d \geq \dim(C_{loc}(\Gamma))$. This new bound is enabled by demonstrating that the set $C_{\text{private}}(\Gamma)$ of correlations using private randomness is connected, allowing the use of an extension of Carathéodory's Theorem. One correlation of this improvement is that every set with $|A| = 1$ or $|B| = 1$ has entanglement dimension 1, and can thus be written as a correlation with private randomness.

In the second part of the thesis we describe a method for the reconstruction of bipartite correlations. This method solves optimization problems for the POVM's and density matrix consecutively, in a so-called see-saw algorithm. It is shown that such problems are semidefinite programs, which allows the use of efficient solvers. By using the algorithm for different dimensions, it is also possible to determine the entanglement dimension of a correlation. One of the strengths of this algorithm is its generality: it applies to different dimensions, question sets and answer sets. Finally, some numerical experiments are performed, demonstrating that the algorithm can reconstruct several correlations with different entanglement dimensions. For higher entanglement dimensions the algorithm is not always able to reconstruct the correlation, possibly due to the computational limitations.

ACKNOWLEDGEMENTS

I would like to thank my review committee for taking the time to help me with this thesis. In particular, I am very grateful to Dr. D. de Laat for providing me with amazing guidance during the process, not only helping me with the research but also making me genuinely excited about my studies. Of course, I am also thankful for the support of my family and friends; especially Billy Verhage, Jort Bouma, Jort de Groot, Rona Roovers and Sarah Jansen have helped me tremendously throughout my studies. Finally, I believe the TU Delft deserves praise for supplying me with interesting and high-quality education over the last three years.

BIBLIOGRAPHY

[Aru+19] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (2019), pp. 505–510.

[Bel64] John S Bell. “On the einstein podolsky rosen paradox”. In: *Physics Physique Fizika* 1.3 (1964), p. 195.

[Bri11] Jop Bri. “Grothendieck inequalities, nonlocal games and optimization”. In: (2011).

[Bru+14] Nicolas Brunner et al. “Bell nonlocality”. In: *Reviews of Modern Physics* 86.2 (2014), p. 419.

[Bun34] Lucas Nicolaas Hendrik Bunt. *Bijdrage tot de theorie der convexe puntverzamelingen*. Rijksuniversiteit te Groningen, 1934.

[Car00] Neal L Carothers. *Real analysis*. Cambridge University Press, 2000, p. 82.

[Fre04] Robert M Freund. “Introduction to semidefinite programming (SDP)”. In: *Massachusetts Institute of Technology* (2004), pp. 8–11.

[GLL18] Sander Gribling, David de Laat, and Monique Laurent. “Bounds on entanglement dimensions and quantum graph parameters via noncommutative polynomial optimization”. In: *Mathematical Programming* 170.1 (2018), pp. 5–42.

[Hen+15] Bas Hensen et al. “Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres”. In: *Nature* 526.7575 (2015), pp. 682–686.

[IIA06] Tsuyoshi Ito, Hiroshi Imai, and David Avis. “Bell inequalities stronger than the Clauser-Horne-Shimony-Holt inequality for three-level isotropic states”. In: *Physical Review A* 73.4 (2006), p. 042109.

[LV12] Monique Laurent and Frank Vallentin. “Semidefinite optimization”. In: *Lecture Notes, available at http://page. mi. fu-berlin. de/fmario/sdp/laurentv. pdf* (2012).

[MB07] Florian Mintert and Andreas Buchleitner. “Observable entanglement measure for mixed quantum states”. In: *Physical review letters* 98.14 (2007), p. 140505.

[PV10] Károly F Pál and Tamás Vértesi. “Maximal violation of a bipartite three-setting, two-outcome Bell inequality using infinite-dimensional quantum systems”. In: *Physical Review A* 82.2 (2010), p. 022116.

[PV16] Carlos Palazuelos and Thomas Vidick. “Survey on nonlocal games and operator space theory”. In: *Journal of Mathematical Physics* 57.1 (2016), p. 015220.

[RV15] Oded Regev and Thomas Vidick. “Quantum XOR games”. In: *ACM Transactions on Computation Theory (ToCT)* 7.4 (2015), pp. 1–43.

[Slo16] William Slofstra. “Tsirelson’s problem and an embedding theorem for groups arising from non-local games (2016)”. In: *arXiv preprint arXiv:1606.03140* (2016).

[Sta18] Cyril J Stark. “Learning optimal quantum models is NP-hard”. In: *Physical Review A* 97.2 (2018), p. 020103.

[VB96] Lieven Vandenberghe and Stephen Boyd. “Semidefinite programming”. In: *SIAM review* 38.1 (1996), pp. 49–95.

[WEH18] Stephanie Wehner, David Elkouss, and Ronald Hanson. “Quantum internet: A vision for the road ahead”. In: *Science* 362.6412 (2018).

[Wil+08] J Wilms et al. “Local realism, detection efficiencies, and probability polytopes”. In: *Physical Review A* 78.3 (2008), p. 032116.

[WW01] Reinhard F Werner and Michael M Wolf. “Bell inequalities and entanglement”. In: *arXiv preprint quant-ph/0107093* (2001).

A

Code

In this Appendix, we give the code that is used to generate the examples in this thesis, as well as some of the underlying functions. It is written in the language Julia, version 1.4.2. Julia can be downloaded online, for example with an IDE such as Juno or VSStudio. To run the code, several packages are needed. The package Mosek requires the installation and set-up of a free license.

The first part of the code entails all relevant functions. In the second part, different sections are commented out. You can un-comment sections to run the code.

The entire file can also be found at GitHub:
https://github.com/JanBosma/Entanglement-Dimension
Feel free to modify the code any way you want.

```
using Convex, Mosek, MosekTools, LinearAlgebra
# Written in Julia Version 1.4.2
# Mosek and Mosektools require a (free) license to be installed.
# Alternatively, one could use the solver SCS

# The code used to obtain the results from the thesis are given below
# If you want to run some code for yourself, you can un-comment a
    ↪ section
# by using CTRL + /

# Variables:
# A = number of answers Alice
# B = number of answers Bob
# S = number of questions Alice
# T = number of questions Bob

# P = given bipartite correlation
# d = entanglement dimension
```

```
# E[s][a] is complex dxd matrix for all s in S and a in A
# F[t][b] is complex dxd matrix for all t in T and b in B
# psi = complex unit vector with dimension d^2
# rho = complex (d^2)x(d^2) matrix with trace 1

function findE(F, rho, P::Array,d,A,B,S,T)
    #Solves SDP for optimal E, given state rho, POVM F and desired P
    E = [[ComplexVariable(d,d) for s=1:S] for a=1:A]
    M = Variable(1, Positive())
    objective = M
    s_1 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
        ↪ ] for a=1:A]
    s_2 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
        ↪ ] for a=1:A]

    constraints = [E[a][s] in :SDP for s=1:S for a=1:A]
    constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho) ==
        ↪  M - s_1[a][b][s][t] for a=1:A for s=1:S for b=1:B for t
        ↪ =1:T]
    constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho) ==
        ↪  -M + s_2[a][b][s][t] for a=1:A for s=1:S for b=1:B for t
        ↪ =1:T]

    ide = Matrix{Float64}(I, d, d)
    for s in 1:S
        constraints += sum(E[a][s] for a=1:A) == ide
    end

    problem = minimize(objective,constraints)
    solve!(problem, () -> Mosek.Optimizer())

    E = [[E[a][s].value for s=1:S] for a=1:A]

end

function findF(E, rho, P::Array,d,A,B,S,T)
    #Solves SDP for optimal F, given density matrix rho, POVM E and
        ↪ desired P
    F = [[ComplexVariable(d,d) for t=1:T] for b=1:B]
    M = Variable(1, Positive())
    objective = M
    s_1 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
        ↪ ] for a=1:A]
    s_2 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
```

```
        ↪ ] for a=1:A]

    constraints = [F[b][t] in :SDP for t=1:T for b=1:B]

    if d==1
        # Of course, this doesn't seem neccessary, but there seems to
            ↪  be
        # a bug with the trace when d=1
        constraints += [P[a][b][s][t] - E[a][s]* F[b][t]*rho == M -
            ↪ s_1[a][b][s][t] for a=1:A for s=1:S for b=1:B for t=1:T
            ↪ ]
        constraints += [P[a][b][s][t] - E[a][s]* F[b][t]*rho == -M +
            ↪ s_2[a][b][s][t] for a=1:A for s=1:S for b=1:B for t=1:T
            ↪ ]

    else
        constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho
            ↪ ) == M - s_1[a][b][s][t] for a=1:A for s=1:S for b=1:B
            ↪ for t=1:T]
        constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho
            ↪ ) == -M + s_2[a][b][s][t] for a=1:A for s=1:S for b=1:B
            ↪  for t=1:T]
    end

    ide = Matrix{Float64}(I, d, d)
    for t in 1:T
        constraints += sum(F[b][t] for b=1:B) == ide
    end

    problem = minimize(objective,constraints)
    solve!(problem, () -> Mosek.Optimizer())

    F = [[F[b][t].value for t=1:T] for b=1:B]
end

function generaterandomrho(d::Int)
    # generates random density matrix
    psi = rand(Float64,(d^2,1)) + im*rand(Float64,(d^2,1))
    psi /= norm(psi, 2)
    rho = psi * psi'
end

function randomE(d,A,S)
    # generates random POVM E, by generating random matrices and
        ↪ solving for
```

```
    # the closest SDP matrices

    E = [[rand(Float64,(d,d))+ im*rand(Float64,(d,d)) for s=1:S] for
        ↪ a=1:A]

    E_SDP= [[ComplexVariable(d,d) for s=1:S] for a=1:A]
    M = Variable(1)
    objective = M

    s_1 = [[Variable(1, Positive()) for s=1:S] for a=1:A]
    s_2 = [[Variable(1, Positive()) for s=1:S] for a=1:A]

    constraints = [E_SDP[a][s] in :SDP for a=1:A for s=1:S]
    constraints += M >= 0

    constraints += [norm(E_SDP[a][s] - E[a][s]) == M - s_1[a][s] for
        ↪ a=1:A for s=1:S]

    constraints += [norm(E_SDP[a][s] - E[a][s]) == -M + s_2[a][s] for
        ↪  a=1:A for s=1:S]

    ide = Matrix{Float64}(I, d, d)
    for s in 1:S
        constraints += sum(E_SDP[a][s] for a=1:A) == ide
    end

    problem = minimize(objective,constraints)
    solve!(problem, () -> Mosek.Optimizer())

# Making sure that all matrices are Hermitian
    E = [[(E_SDP[a][s].value+E_SDP[a][s].value')/2 for s=1:S] for a
        ↪ =1:A]

end

function generaterandomoperators(d,A,B,S,T)
    rhostart = generaterandomrho(d)
    ide = Matrix{Float64}(I, d, d)

    Estart = randomE(d,A,S)
    Fstart = randomE(d,B,T)
    rhostart, Estart, Fstart
end

function generateP(rho, E, F,d,A,B,S,T)
```

```
    [[[[tr(kron(E[a][s], F[b][t])*rho) for t=1:T] for s=1:S] for b=1:
        ↪ B] for a=1:A]
end

function testfindE(d,A,B,S,T)
    rho, E, F= generaterandomoperators(d,A,B,S,T)
    P = generateP(rho, E, F,d,A,B,S,T)
    findE(F, rho, P,d,A,B,S,T)
end

function findrho(E,F,P,d,A,B,S,T)
    # solves SDP for optimal density matrix rho, given POVM's E and F
        ↪ ,
    # and desired correlation P
    rho = ComplexVariable(d^2,d^2)
    M = Variable(1, Positive())
    objective = M
    s_1 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
        ↪ ] for a=1:A]
    s_2 = [[[[Variable(1, Positive()) for t=1:T] for s=1:S] for b=1:B
        ↪ ] for a=1:A]

    constraints = rho in :SDP
    constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho) ==
        ↪  M - s_1[a][b][s][t] for a=1:A for s=1:S for b=1:B for t
        ↪ =1:T]
    constraints += [P[a][b][s][t] - tr(kron(E[a][s], F[b][t])*rho) ==
        ↪  -M + s_2[a][b][s][t] for a=1:A for s=1:S for b=1:B for t
        ↪ =1:T]
    constraints += tr(rho) == 1

    problem = minimize(objective,constraints)
    solve!(problem, () -> Mosek.Optimizer())
    rho.value
end

function randomP_fixed_d(num_iterations,d,A,B,S,T)
    #executes see-saw algorithm of num_iterations iterations
    #returns a list of errors per step
    rho, E, F = generaterandomoperators(d,A,B,S,T)
    P_real = generateP(rho, E, F,d,A,B,S,T) # We want to approximate
        ↪ this P
    errors = zeros(0)
    rho, E, F = generaterandomoperators(d,A,B,S,T) # We initialize
        ↪ with new rho,E,F
```

```
    for i in 1:num_iterations
            myerror = maximum(norm(P_real[a][b][s][t] - generateP(rho,
                ↪ E,F,d,A,B,S,T)[a][b][s][t]) for t=1:T for s=1:S for
                ↪  b=1:B for a=1:A)
            push!(errors,myerror)
        E = findE(F,rho,P_real,d,A,B,S,T)
            myerror = maximum(norm(P_real[a][b][s][t] - generateP(rho,
                ↪ E,F,d,A,B,S,T)[a][b][s][t]) for t=1:T for s=1:S for
                ↪  b=1:B for a=1:A)
            push!(errors,myerror)
        F = findF(E,rho,P_real,d,A,B,S,T)
            myerror = maximum(norm(P_real[a][b][s][t] - generateP(rho,
                ↪ E,F,d,A,B,S,T)[a][b][s][t]) for t=1:T for s=1:S for
                ↪  b=1:B for a=1:A)
            push!(errors,myerror)
        rho = findrho(E,F,P_real,d,A,B,S,T)
    end

    errors

end

function randomP_increasing_d(num_iterations,d,d_max,epsilon,A,B,S,T)
    #The complete algorithm, determining the correlation and the
        ↪ entanglement
    # dimension
    # This version generates a P itself
    # In most cases, a solution is found for samples =1. However,
        ↪ there have
    # been instances where samples was higher.
    # Sometimes, here is a bug, in which the POVM's vanish, causing
        ↪ an error
    rho, E, F = generaterandomoperators(d,A,B,S,T)
    P_real = generateP(rho, E, F,d,A,B,S,T) # We want to approximate
        ↪ this P
    d=1
    errors = zeros(0)
    while d<d_max+1
        for sa in 1:samples
            errors = zeros(0)
            rho, E, F = generaterandomoperators(d,A,B,S,T) # We
                ↪ initialize with new rho,E,F
            for i in 1:num_iterations
                    myerror = maximum(norm(P_real[a][b][s][t] -
                        ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
```

```
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
                println(errors)
            E = findE(F,rho,P_real,d,A,B,S,T)
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
                println(errors)
            F = findF(E,rho,P_real,d,A,B,S,T)
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
                println(errors)
            rho = findrho(E,F,P_real,d,A,B,S,T)
            if myerror < epsilon
                return errors, d,sa,E,F,"Success"
            end
        end
    end
    d+=1
end
return errors,"Failed"
end

function givenP_increasing_d(P_real,num_iterations,d_max,epsilon,A,B,
    ↪ S,T)
    #The complete algorithm, determining the correlation and the
        ↪ entanglement
    # dimension
    # This version takes the correlation as input
    # In most cases, a solution is found for samples =1. However,
        ↪ there have
    # been instances where samples was higher.
    # There is a bug, in which the POVM's vanish, causing an error

    d=1
    errors = zeros(0)
    while d<d_max+1
        for sa in 1:samples
            errors = zeros(0)
            rho, E, F = generaterandomoperators(d,A,B,S,T) # We
                ↪ initialize with new rho,E,F
            for i in 1:num_iterations
```

```
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
            E = findE(F,rho,P_real,d,A,B,S,T)
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
            F = findF(E,rho,P_real,d,A,B,S,T)
                myerror = maximum(norm(P_real[a][b][s][t] -
                    ↪ generateP(rho,E,F,d,A,B,S,T)[a][b][s][t])
                    ↪ for t=1:T for s=1:S for b=1:B for a=1:A)
                push!(errors,myerror)
            rho = findrho(E,F,P_real,d,A,B,S,T)
            if myerror < epsilon
                return d,errors,E,F,rho,generateP(rho,E,F,d,A,B,S,T
                    ↪ ),sa,"Success"
            end
        end
    end
    d+=1
end
return errors,"Failed"
end

################ Deterministic Correlations ################

# A=2
# B=2
# S=2
# T=2
# num_iterations = 20
# samples = 5
# d=1
# d_max = 4
# epsilon = 10^-8
#
# P_det = [
# [
# [[0,0],[0,0]],
# [[0,0],[0,0]]
# ],
# [
# [[0,0],[0,0]],
```

```
# [[0,0],[0,0]]
# ]
# ]
# P_det[1][1][1][1]=1
# P_det[1][2][1][2]=1
# P_det[2][1][2][1]=1
# P_det[2][2][2][2]=1
#
# P_used = P_det
#
# println("Start")
# println("====================================")
# println(givenP_increasing_d(P_used,num_iterations,d_max,epsilon,A,B
    ↪ ,S,T))

############# Private Randomness Correlations ######################

# A=4
# B=4
# S=2
# T=2
# num_iterations = 10
# samples = 2
# d_max = 4
# epsilon = 10^-8
#
# function P_A_priv(a,s)
# if a==1 && s==1
# return 0.5
# elseif a==2 && s==1
# return 0.5
# elseif a==3 && s==2
# return 0.5
# elseif a==4 && s==2
# return 0.5
# else
# return 0
# end
# end
#
# function P_B_priv(b,t)
# if b==1 && t==1
# return 1/4
# elseif b==2 && t==1
# return 3/4
```

```
# elseif b==3 && t==2
# return 0.1
# elseif b==4 && t==2
# return 0.9
# else
# return 0
# end
# end
#
#
# P_pcorr = [[[[P_A_priv(a,s)*P_B_priv(b,t) for t=1:T] for s=1:S] for
    ↪  b=1:B] for a=1:A]
#
# println("Start")
# println("====================================")
# println(givenP_increasing_d(P_pcorr,num_iterations,d_max,epsilon,A,
    ↪ B,S,T))

############Approximated Shared Randomness Correlations
    ↪ ##################3

# A=2
# B=13
# S=2
# T=2
# num_iterations = 10
# samples = 2
# d_max = 4
# epsilon = 10^-3
#
# #the following empirical result is found using Python, with 10^6
    ↪ samples.
# python = [[[[0.250679, 0.0], [0.041953, 0.0]], [[0.0, 0.0],
    ↪ [0.083058, 0.0]], [[0.249778, 0.014256], [0.041774,
    ↪ 0.027882]], [[0.0, 0.027541], [0.0, 0.0]], [[0.0, 0.041567],
    ↪ [0.0, 0.027583]], [[0.0, 0.05552], [0.0, 0.0]], [[0.0,
    ↪ 0.069334], [0.0, 0.027813]], [[0.0, 0.083663], [0.0, 0.0]],
    ↪ [[0.0, 0.069553], [0.0, 0.027837]], [[0.0, 0.055741], [0.0,
    ↪ 0.0]], [[0.0, 0.041969], [0.0, 0.02744]], [[0.0, 0.027966],
    ↪ [0.0, 0.0]], [[0.0, 0.014026], [0.0, 0.027884]]], [[[0.0,
    ↪ 0.0], [0.20881, 0.0]], [[0.500309, 0.0], [0.41673, 0.0]],
    ↪ [[0.0, 0.014123], [0.208567, 0.0]], [[0.0, 0.027882], [0.0,
    ↪ 0.055707]], [[0.0, 0.041501], [0.0, 0.055517]], [[0.0,
    ↪ 0.055349], [0.0, 0.11076]], [[0.0, 0.069946], [0.0,
    ↪ 0.111288]], [[0.0, 0.08325], [0.0, 0.166194]], [[0.0,
```

```
    ↪ 0.069919], [0.0, 0.111335]], [[0.0, 0.055449], [0.0,
    ↪ 0.110674]], [[0.0, 0.041436], [0.0, 0.055508]], [[0.0,
    ↪ 0.02827], [0.0, 0.055629]], [[0.0, 0.013967], [0.0, 0.0]]]]
#
# println("Start")
# println("====================================")
# println(givenP_increasing_d(python,num_iterations,d_max,epsilon,A,B
    ↪ ,S,T))


################## Example Shared Randomness Correlations
    ↪ ####################
#
# A=2
# B=3
# S=2
# T=2
# num_iterations = 5
# samples = 1
# d_max = 7#S*(A-1)+T*(B-1)+S*T*(A-1)*(B-1)
# epsilon = 10^-4
#
# P_shar = [[[[0.0 for t=1:T] for s=1:S] for b=1:B] for a=1:A]
#
# P_shar[1][1][1][1]=0.25
# P_shar[1][1][1][2]=0.5*5/6*5/6
# P_shar[1][1][2][1]=1/4*1/6
# P_shar[1][1][2][2]=5/36
# P_shar[1][2][1][1]=0.0
# P_shar[1][2][1][2]=0.5*2*1/6*5/6
# P_shar[1][2][2][1]=0.5*1/6
# P_shar[1][2][2][2]=0.0
# P_shar[1][3][1][1]=1/4
# P_shar[1][3][1][2]=0.5*1/36
# P_shar[1][3][2][1]=0.25*1/6
# P_shar[1][3][2][2]=1/36
# P_shar[2][1][1][1]=0.0
# P_shar[2][1][1][2]=0.5*5/6*5/6
# P_shar[2][1][2][1]=1/4*5/6
# P_shar[2][1][2][2]=5/6*4/6
# P_shar[2][2][1][1]=1/2
# P_shar[2][2][1][2]=0.5*2*1/6*5/6
# P_shar[2][2][2][1]=0.5*5/6
# P_shar[2][2][2][2]=2*1/6*5/6
# P_shar[2][3][1][1]=0.0
```

```
# P_shar[2][3][1][2]=0.5*1/36
# P_shar[2][3][2][1]=5/6*1/4
# P_shar[2][3][2][2]=0.0
#
# println("Start")
# println("====================================")
# println(givenP_increasing_d(P_shar,num_iterations,d_max,epsilon,A,B
    ↪ ,S,T))


################ Given Shared Randomness correlation ##############

# A=2
# B=2
# S=2
# T=2
# num_iterations = 20
# samples = 2
# d_max = S*(A-1)+T*(B-1)+S*T*(A-1)*(B-1)
# epsilon = 10^-8

# P_shar = [[[[0.0 for t=1:T] for s=1:S] for b=1:B] for a=1:A]
#
# P_shar[1][1][1][1]=0.25
# P_shar[1][1][1][2]=0.5*5/6*5/6
# P_shar[1][1][2][1]=1/4*1/6
# P_shar[1][1][2][2]=5/36
# P_shar[1][2][1][1]=1/4
# P_shar[1][2][1][2]=0.5*2*1/6*5/6 + 0.5*1/36
# P_shar[1][2][2][1]=0.5*1/6 + 0.25*1/6
# P_shar[1][2][2][2]=1/36
# P_shar[2][1][1][1]=0.0
# P_shar[2][1][1][2]=0.5*5/6*5/6
# P_shar[2][1][2][1]=1/4*5/6
# P_shar[2][1][2][2]=5/6*4/6
# P_shar[2][2][1][1]=1/2
# P_shar[2][2][1][2]=0.5*2*1/6*5/6 + 0.5*1/36
# P_shar[2][2][2][1]=0.5*5/6+5/6*1/4
# P_shar[2][2][2][2]=2*1/6*5/6


# println("Start")
# println("====================================")
# println(givenP_increasing_d(P_shar,num_iterations,d_max,epsilon,A,B
    ↪ ,S,T))
```

```
################## Testing randomly generated classical correlations
    ↪ #########

# function randomshared(A,B,S,T,dets)
# # Generates random classical correlations (C_loc(Gamma))
# P_shar2 = [[[[[0.0 for t=1:T] for s=1:S] for b=1:B] for a=1:A] for
    ↪ i=1:dets]
# lambda=rand(Float64,(dets,1))
# sumlambda = 0.0
# for i=1:dets
# sumlambda += lambda[i]
# end
# for i=1:dets
# lambda[i]/=sumlambda
# end
# for i=1:dets
# for s=1:S
# for t= 1:T
# a=rand((1,A))
# b=rand((1,B))
# P_shar2[i][a][b][s][t]=1
# end
# end
# end
#
# result = [[[[0.0 for t=1:T] for s=1:S] for b=1:B] for a=1:A]
# for t=1:T
# for s=1:S
# for b=1:B
# for a=1:A
# res=0.0
# for i=1:dets
# res+=lambda[i]*P_shar2[i][a][b][s][t]
# end
# result[a][b][s][t]=res
# end
# end
# end
# end
#
# result = [[[[result[a][b][s][t] for t=1:T] for s=1:S] for b=1:B]
    ↪ for a=1:A]
# end
#
```

```
# A=2
# B=2
# S=2
# T=2
# num_iterations = 20
# samples = 2
# d_max = S*(A-1)+T*(B-1)+S*T*(A-1)*(B-1)
# epsilon = 10^-4
# dets=5
#
# P_shar3 = randomshared(A,B,S,T,dets)
#
# println("Start")
# println("====================================")
# println(givenP_increasing_d(P_shar3,num_iterations,d_max,epsilon,A,
    ↪ B,S,T))

############ Quantum correlations ###################

A=1
B=2
S=3
T=4
d=3
num_iterations = 20
samples = 5
d_max = d+1
epsilon = 10^-8

println("Start")
println("====================================")
println(randomP_increasing_d(num_iterations,d,d_max,epsilon,A,B,S,T))
```